FABRYKA INŻYNIERÓW

# WIZUALIZACJA DANYCH BIOMEDYCZNYCH

Mirosław Socha

CZŁOWIEK – NAJLEPSZA INWESTYCJA

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Wydawnictwa Akademii Górniczo-Hutniczej im. Stanisława Staszica w Krakowie

Redaktor Naczelny Wydawnictw AGH: *Jan Sas*

Komitet Naukowy Wydawnictw AGH:
*Tomasz Szmuc* (przewodniczący), *Marek Capiński*, *Jerzy Klich*, *Witold K. Krajewski*, *Tadeusz Sawik*, *Mariusz Ziółko*

Recenzenci: *prof. dr hab. inż. Andrzej Materka*
*dr hab. inż. Krzysztof Boryczko, prof. nadzw. AGH*

Autor pracuje w Katedrze Metrologii
na Wydziale EAIiE Akademii Górniczo-Hutniczej w Krakowie

Redakcja: *Małgorzata Koch*

Projekt okładki serii: *Barbara Jezierska*
Przygotowanie do druku okładki i stron tytułowych: *Zofia Łucka*


ISBN 978-83-7464-376-4

Publikacja współfinansowana przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

Skład komputerowy: „Edycja”

Egzemplarz bezpłatny

Redakcja Wydawnictw AGH
al. Mickiewicza 30, 30-059 Kraków
tel. 12 617 32 28, tel./faks 12 636 40 38
e-mail: redakcja@wydawnictwoagh.pl
www.wydawnictwa.agh.edu.pl

# Spis treści

# Wstęp

Wizualizacja danych rozumiana jako przedstawianie informacji w postaci graficznej jest obecnie bardzo szeroko stosowana. Trudno sobie wyobrazić dzisiejszy przemysł, medycynę, naukę czy też szkolnictwo bez takich przekazów graficznych jak: rysunki, wykresy, schematy czy diagramy. Techniki graficznego przedstawiania informacji są stosowane od zarania dziejów, jednak to rozwój komputerów spowodował gwałtowny rozwój zupełnie nowych metod przetwarzania i prezentacji danych. Nowe techniki pomiarowe czy też zaawansowane symulacje dostarczają obecnie ogromnych ilości danych, których analiza i interpretacja bez odpowiedniego przetworzenia i prezentacji byłaby utrudniona, a często nawet niemożliwa. Dobrze zaprojektowana i wykonana wizualizacja, szczególnie w zastosowaniu do danych medycznych, może dostarczyć bezcennych informacji, np. decydujących o szansach na przeżycie pacjenta. Sposób postrzegania oraz odbioru obrazu przez człowieka jest procesem bardzo złożonym. Zdaniem autora, sama znajomość technik prezentacji danych może być niewystarczająca do tworzenia wizualizacji zrozumiałych dla odbiorcy. Dlatego też w niniejszej pracy zawarto informacje na temat percepcji obrazu przez człowieka. Na postrzeganie oraz interpretację obrazu ma wpływ budowa zmysłu wzroku, sposób przetwarzania bodźców wzrokowych, jak również uwarunkowania psychologiczne, a nawet kulturowe. W pracy omówione zostaną również najczęściej spotykane formaty zapisu danych, w tym standard DICOM służący do przechowywania danych medycznych oraz zyskujący na popularności hierarchiczny format danych HDF. Na przykładzie biblioteki *Visualization Toolkit* (VTK) oraz programu *Matlab*® przedstawione zostaną konkretne techniki wizualizacji danych.

Niniejsza książka kierowana jest do studentów ostatnich lat studiów technicznych, głównie programistów, którzy pragną poszerzyć swoją wiedzę na temat metod i narzędzi informatycznych, umożliwiających wizualizację danych. Osoby niezainteresowane programowaniem znajdą w pracy informacje na temat stosowanych programów, które pozwolą praktycznie wykorzystać przedstawioną wiedzę teoretyczną.

Celem pracy jest przekazanie wiedzy na temat sposobu postrzegania oraz interpretacji obrazu przez człowieka, zasad tworzenia poprawnych przekazów graficznych oraz posługiwania się podstawowymi i zaawansowanymi metodami prezentacji danych, zwłaszcza w odniesieniu do danych medycznych.

Książka podzielona jest na pięć rozdziałów. Pierwszy zawiera informacje wprowadzające dotyczące wizualizacji danych, takie jak: główne obszary zastosowań, definicje pojęć związanych z wizualizacją oraz przepływ informacji między jej poszczególnymi etapami. Ponieważ

wizualizacja danych głównie opiera się na zmyśle wzroku, omówione zostały podstawy percepcji obrazu. W kolejnym podrozdziale, przedstawiono najczęściej stosowane modele barw ich główne zastosowania oraz ograniczenia. Omówiono również zagadnienia związane z zarządzaniem informacją o kolorze. Ostatni podrozdział dotyczy zdolności analizy obrazu przez człowieka. Przedstawiono w nim, opracowane przez psychologów, tzw. *prawa* teorii percepcji wzrokowej, które opisują, w jaki sposób człowiek postrzega i interpretuje obraz. Znajomość tych praw pozwala przewidzieć, w jaki sposób obraz może zostać zinterpretowany.

W rozdziale drugim przedstawione zostały podstawy cyfrowej reprezentacji danych, zdefiniowano niezbędne pojęcia, omówiono podstawowe typy informacji oraz metody ich cyfrowego kodowania. W kolejnych podrozdziałach przedstawiono cztery formaty danych: bardzo prosty i uniwersalny format CSV, formaty używane przez bibliotekę do wizualizacji danych *Visualization Toolkit* (VTK), standard DICOM do przechowywania danych medycznych oraz uniwersalny, hierarchiczny format danych HDF. W każdym rozdziale znajdują się informacje wprowadzające, kierowane głównie do „użytkowników" plików oraz bardziej szczegółowe, kierowane do programistów.

Rozdział trzeci zawiera opis biblioteki *Visualization Toolkit* (VTK), która jest specjalizowanym narzędziem informatycznym do przetwarzania i wizualizacji danych. W rozdziale tym omówiono architekturę biblioteki: budowę strumienia wizualizacji, zastosowany model graficzny odpowiadający za graficzną prezentację danych oraz jego podstawowe obiekty. W kolejnych podrozdziałach omówione są zagadnienia związane z zarządzaniem pamięcią, sposobem reprezentacji danych, typami i metodami interpolacji danych. Następnie omówione zostały metody proceduralnego tworzenia podstawowych reprezentacji danych, obiekty służące do zapisy i odczytu danych oraz wizualizacji. Dalej przedstawione zostały mechanizmy odpowiedzialne za wymianę komunikatów między obiektami biblioteki VTK. W ostatnim podrozdziale omówiono metody wykorzystywania biblioteki VTK w aplikacjach tworzonych w językach skryptowych oraz w C++. Przedstawiono również przegląd programów, które umożliwiają graficzne programowanie aplikacji do przetwarzania i wizualizacji danych.

Rozdział czwarty omawia podstawowe metody wizualizacji danych. Na początku omówiono zasady tworzenia przekazu graficznego w oparciu o semiologię grafiki, naukę badającą wpływ znaków graficznych na porozumiewanie się ludzi. Zaprezentowane zostały parametry charakteryzujące przetwarzane dane, zdefiniowano zmienne wizualne oraz sposoby ich skutecznego mapowania. W podsumowaniu tego podrozdziału znajduje się przegląd metod wizualnej prezentacji danych w postaci: diagramów, sieci, map oraz symboli. Przegląd uwzględnia liczbę składowych, sposób organizacji znaków graficznych oraz rodzaje mapowań. W kolejnym podrozdziale przybliżono możliwości graficzne programu Matlab, zwłaszcza funkcji odpowiedzialnych za dwuwymiarową oraz trójwymiarową wizualizacją. Przytoczono również przykład skryptu, który przedstawia sposób modyfikacji parametrów obiektów graficznych z poziomu kodu. Następne podrozdziały opisują podstawowe algorytmy wizualizacji danych skalarnych oraz wektorowych. W przypadku danych skalarnych omówione zostały algorytmy mapowania kolorów oraz konturowania. Opisy obejmują podstawy teoretyczne oraz przykłady implementacji algorytmów, dostępnych zarówno w postaci funkcji programu Matlab, jak i odpowiednich klas biblioteki VTK. Dla danych wektorowych zaprezentowano cztery podstawowe techniki: wizualizację przy użyciu znaczników, metody

bazujące na zmianie geometrii obiektów przez ich „wyginanie", wizualizację pól wektorowych w postaci linii prądów oraz technikę elipsoidy tensorowej.

W ostatnim, piątym rozdziale zaprezentowano zaawansowane algorytmy wizualizacji danych, które są powszechnie wykorzystywane w wizualizacjach danych medycznych. Są to metody umożliwiające tworzenie obrazów (rendering) oświetlonych powierzchni oraz rendering objętościowy, inaczej wolumetryczny. Ponieważ obydwie metody wizualizacji bardzo silnie wykorzystują nowoczesne karty graficzne do akceleracji obliczeń, w rozdziale tym zamieszczono krótki rys historyczny rozwoju kart graficznych oraz omówiono podstawowe etapy pracy karty graficznej, w wyniku których powstaje obraz wirtualnej, trójwymiarowej sceny. Następnie przybliżone zostały zagadnienia związane z możliwościami programowalnego potoku graficznego, modelowania oświetlenia oraz metod interpolacji kolorów. Ostatnia część rozdziału dotyczy metod renderingu objętościowego (wolumetrycznego). Omówiono w nim podstawy teoretyczne tej metody, przytoczono równanie renderingu wolumetrycznego oraz omówiono strumień wizualizacji wolumetrycznej wraz z opisem jego poszczególnych kroków. Następnie, przedstawiono podstawowe algorytmy renderingu wolumetrycznego oraz podstawy doboru funkcji przejścia. Rozdział zakończony jest omówieniem przykładowego programu, który wykorzystuje obiekty biblioteki VTK do wizualizacji objętościowej gęstości ładunku wokół cząsteczki.

Na końcu pracy umieszczono dodatki, które są uzupełnieniem powyższych rozdziałów, a są to: bibliografia, spis kodów źródłowych oraz skorowidz.

Niniejszy podręcznik stanowi jedynie wprowadzenie do zagadnień związanych z wizualizacją danych. Wiele bardzo ciekawych technik, które znajdują zastosowanie w prezentacji danych medycznych, zostało jedynie wspomnianych lub nie zostało omówionych. Można do nich zaliczyć między innymi rendering niefotorealistyczny (np. ilustracyjny), którego głównym zadaniem jest wzmocnienie percepcji wybranych cech trójwymiarowego kształtu, np. wydobycie informacji na temat krawędzi. W przypadku renderingu wolumetrycznego, bardzo szerokie możliwości kształtowania wyników wizualizacji umożliwiają różne warianty tzw. funkcji przejścia. Funkcje te mogą być jedno- oraz wielowymiarowe, mogą uwzględniać dodatkowe informacje np. odległość lub gradient danych, znacznie poszerzając i ułatwiając percepcję informacji. Coraz bardziej dostępne stają się również badania diagnostyczne z wykorzystaniem funkcyjnego rezonansu magnetycznego (fMRI). Wyniki tych badań, w postaci zmiennych w czasie, przestrzennych pól tensorowych wymagają zastosowania rozbudowanych algorytmów do wyznaczania oraz grupowania tzw. linii prądów, czy też zaawansowanych metod projektowania i rozmieszczania znaczników.

Bardzo dużym wyzwaniem jest analiza i wizualizacja danych medycznych o znacznej liczbie wymiarów, których liczba może sięgać nawet kilku tysięcy. Przykładem takich danych są wyniki eksperymentów z wykorzystaniem mikromacierzy: analizy DNA, badania ekspresji lub sekwencji genów, badania biologiczne (np. białek), medyczne (np. tkanek) czy też chemiczne (analiza związków chemicznych). Innym przykładem mogą być wyniki symulacji biochemicznych, czy też pomiary z wielokanałowych sond badających żyjące tkanki. Tak skomplikowane dane pomiarowe wymagają zastosowania wstępnego przetwarzania, np. polegającego na grupowaniu (np. k-średnich), redukcji wymiarów (np. poprzez analizę głównych składowych, algorytm PCA) czy też dekompozycji danych (np. metodą SVD).

Do ich wizualizacji opracowano dedykowane techniki prezentacji graficznej danych o znacznej liczbie wymiarów, np. wykresy o równoległych osiach (ang. *parallel coordinates plot*), metody oparte o projekcję wielowymiarowych danych na płaszczyznę, macierze dwuwymiarowych wykresów (ang. *scatter plot matrix, trellis graphs*) czy też wizualizacje macierzy (ang. *matrix visualization*), która nie wymaga redukcji wymiaru danych i polega na odpowiedniej reorganizacji danych źródłowych, a następnie ich „pokolorowaniu".

Szybko rozwijającą się dziedziną nauki jest również wizualizacja informacji, która może być jednym z etapów eksploracji danych (ang. *data mining*). Techniki te, umożliwiają znajdowanie ukrytych w „bazach danych" (tzw. hurtowniach danych, ang. *data warehouse*) prawidłowości lub zależności. Wykorzystanie technik wizualizacji informacji staje się coraz łatwiejsze, np. dzięki rozszerzeniu *Titan Informatics Toolkit*, które znacznie poszerzyło możliwości wizualizacji biblioteki *Visualization Toolkit.*

Autor ma nadzieję, że powyższe braki zostaną uzupełnione w kolejnym wydaniu podręcznika.

Jeśli jeden obraz wart jest tysiąca słów,
czy tysiąc słów może wyjaśnić jeden obraz?
*Michael Stephan*

# Rozdział 1
# Wprowadzenie do wizualizacji danych

Popularne przysłowie chińskie mówi, że *jeden obraz wart jest tysiąca słów.* I trudno się z tym powiedzeniem nie zgodzić. Współczesna cywilizacja, nazywana coraz częściej „cywilizacją obrazkową", bardzo chętnie wykorzystuje przekaz wizualny. Obrazy stanowią nieodłączną część naszej rzeczywistości. Na każdym kroku spotykamy się z informacją wizualną: wykresy i diagramy w telewizji oraz prasie, znaki drogowe, piktogramy oznaczające np. aptekę bądź restaurację, oznaczenia funkcji na sprzęcie elektronicznym itp. Tekst jest ograniczany do minimum i często stanowi jedynie dodatek. Grafika stała się „językiem uniwersalnym", rozumianym niezależnie od kraju czy kultury.

Trudno sobie wyobrazić dzisiejszy przemysł, medycynę, naukę czy szkolnictwo bez wizualizacji: wykresów, schematów, diagramów itp. Dlaczego tak się stało? Prawdopodobnie dlatego, że ludzie w zdecydowanej większości są „wzrokowcami" i bardzo silnie reagują na bodźce wzrokowe. Zmysł wzroku dostarcza ogromnych ilości danych. Jest to spowodowane tym, że w procesie „patrzenia" uczestniczą rozległe obszary mózgu odpowiedzialne za odbiór sygnałów z oczu, a następnie ich interpretację [War04]. Obraz jest najpierw rozkładany na podstawowe składniki: kolor, krawędzie, położenie oraz ruch, a następnie z powrotem syntezowany z uwzględnieniem wspomnień, emocji oraz informacji z innych zmysłów. To, w jaki sposób potrafimy zinterpretować widziany obraz, zależy od wytrenowania mózgu. Ten sam obraz może zostać skrajnie różnie zinterpretowany przez różne osoby, a nawet przez tę samą osobę, ale w innych warunkach. Czy jest więc możliwe, parafrazując pytanie Michaela Stephana, by przygotowany dla konkretnego odbiorcy przekaz wizyjny, mógł dostarczyć „tysiąca właściwych słów"?

Wizualizacja danych rozumiana jako przedstawianie informacji w postaci graficznej, takiej jak: obrazy, diagramy lub animacje, jest dyscypliną nauki łączącą różne dziedziny wiedzy. Podstawowym narzędziem wizualizacji jest grafika, w ostatnich latach silnie wspierana przez grafikę komputerową. Do tworzenia *dobrych wizualizacji*, czyli zrozumiałych dla odbiorcy, niezbędna jest wiedza na temat percepcji obrazów przez człowieka oraz technik i metod prezentacji obrazów. Niezbędna jest również wiedza na temat sposobu pozyskiwania oraz przetwarzania wizualizowanych danych. Dodatkowo, jeżeli danymi są informacje medyczne, bardzo przydatna staje się specjalistyczna wiedza medyczna. Wizualizacja danych jest więc narzędziem, które umożliwia prezentację wyników pomiarów, a także ułatwia ich analizę. Możliwe jest również wykorzystanie rezultatów wizualizacji do testowania oraz weryfikacji hipotez, stawianych na podstawie wyników pomiarów i symulacji.

Wizualizacja danych w medycynie wykorzystywana jest coraz powszechniej, zwłaszcza w następujących dziedzinach [PB07]:

**Edukacja** – przestrzenna wizualizacja organów wewnętrznych stanowi podstawę w informatycznych systemach wspomagających naukę anatomii czy też w systemach przystosowanych do szkolenia i trenowania zabiegów, np. operacji neurologicznych. Wizualizacje umożliwiają analizę wzajemnych relacji przestrzennych między organami, a także umożliwiają dokładną analizę cech morfologicznych.

**Diagnostyka** – na podstawie interaktywnych wizualizacji danych, głównie radiologicznych, możliwe jest np. dokonywanie pomiarów zarówno geometrii, jak i parametrów tkanek (np. gęstości kości). Możliwe jest łączenie danych tego samego pacjenta uzyskanych z różnych źródeł, np. pochodzących z tomografii komputerowej oraz rezonansu magnetycznego, co daje zupełnie nowe możliwości diagnostyczne lekarzom. Coraz częściej stosuje się tzw. zabiegi wirtualne, np. wirtualną kolonoskopię – zabieg zdecydowanie mniej nieprzyjemny dla pacjenta, a jednocześnie umożliwiający wykrycie polipów w jelicie grubym [SSZD07].

**Planowanie zabiegów oraz terapii** – interaktywna, przestrzenna wizualizacja danych medycznych umożliwia precyzyjne zaplanowanie zabiegów, np. skomplikowanych zabiegów chirurgicznych. Dzięki możliwości łączenia informacji z różnych źródeł (dane medyczne i symulacje), możliwe jest np. wspieranie planowania radioterapii.

**Wspomaganie podczas zabiegów** – coraz częstsze jest stosowanie specjalistycznych systemów wspomagających lekarza nie tylko przed zabiegiem w czasie planowania, ale również podczas samych zabiegów [MJM+10, TZD+06, BDD+07].

W niniejszym podręczniku zaprezentowane zostaną techniki umożliwiające głównie wizualizację danych medycznych, które z powodzeniem mogą być zastosowane w innych dziedzinach. Na przykładzie biblioteki *Visualization Toolkit* (VTK) przedstawione zostaną konkretne techniki przetwarzania i prezentacji danych.

Istnieje szereg pojęć związanych z wizualizacją i grafiką komputerową, które w języku potocznym często są błędnie stosowane zamiennie. Dla usystematyzowania nazewnictwa zostaną przytoczone za Schroederem [SML04] definicje pojęć związanych z tworzeniem i przetwarzaniem grafiki komputerowej:

**Przetwarzanie obrazów** – techniki umożliwiające transformowanie obrazów (np. rotacja, skalowanie itp.), poprawę jakości obrazów (np. odszumianie), analizę oraz wydobywanie informacji z obrazów (np. poprzez zliczanie obiektów). Zazwyczaj przetwarzanie obrazów odnosi się do danych dwuwymiarowych, ale może być rozszerzone do innych typów danych.

**Grafika komputerowa** – to techniki (wykorzystujące sprzęt) i metody (algorytmy) tworzenia obrazów z wykorzystaniem komputera. Obejmują zarówno techniki tworzenia obrazów dwuwymiarowych (2D z ang. *two-dimensional*), oparte na klasycznych metodach tworzenia rysunków (np. rysowanie linii na ekranie monitora), jak i techniki tworzenia obrazów przestrzennych, trójwymiarowych (ang. *rendering 3D* ). Obrazy trójwymiarowe mogą powstawać jako projekcje przestrzennych scen na płaszczyznę obrazu, które

są następnie prezentowane na wydrukach lub monitorach. Coraz częściej, dzięki rozwojowi technologii wyświetlaczy 3D (ang. *three-dimensional*), wizualizacje mogą być również prezentowane jako obrazy wywołujące wrażenie przestrzennych, czyli takich, w których mózg wykrywa głębię obrazu (np. postrzegane są wzajemne relacje między obiektami).

**Wizualizacja** – to techniki umożliwiające przeglądanie, analizę, transformowanie oraz prezentacje danych w postaci obrazów (informacji wizyjnej). Głównym celem jest w tym wypadku zwiększenie możliwości zrozumienia informacji zawartych w prezentowanych danych, poprawa czytelności, np. poprzez prezentację w sposób bardzie intuicyjny, bardziej spójny lub z pominięciem informacji zbędnych.

**Interfejs użytkownika** – to cześć systemu informatycznego (oprogramowania wraz ze sprzętem) umożliwiająca intuicyjne sterowanie oprogramowaniem np. do wizualizacji danych. Stanowi pomost między użytkownikiem a zazwyczaj skomplikowanymi algorytmami. Dobrze zaprojektowany interfejs użytkownika ułatwia analizę oraz zwiększa skuteczność wizualizacji.

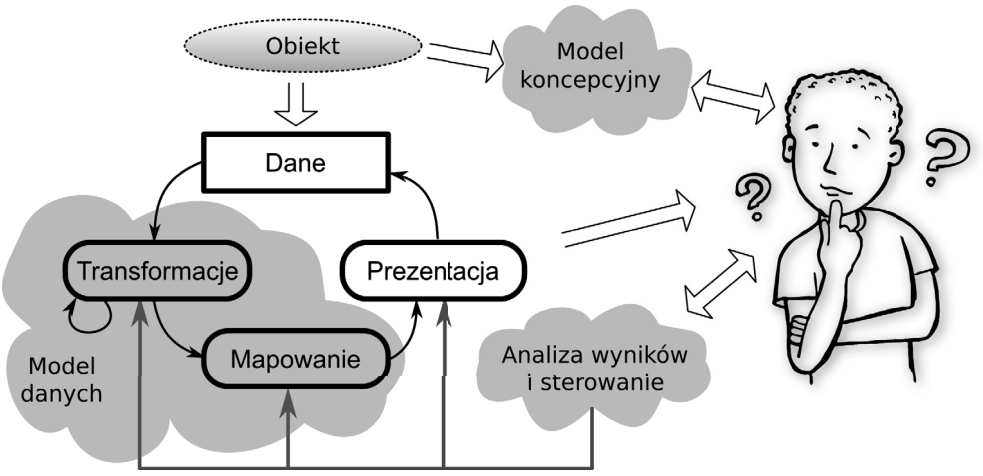
Jak można zauważyć, powyższe dziedziny wzajemnie się zazębiają, przenikają i uzupełniają. *Wizualizacja* sterowana poprzez *interfejs użytkownika* używa *przetwarzania obrazów* do wydobywania informacji, zaś *grafika komputerowa* udostępnia narzędzi do tworzenia obrazów, stanowi wyjście dla wyników działania algorytmów wizualizacji. Do zrozumienia zawartości powstających w trakcie wizualizacji obrazów niezbędna jest wiedza, jak te obrazy należy interpretować. Dlatego też wizualizacja danych zawsze powinna być zaprojektowana dla konkretnego rodzaju danych, z uwzględnieniem zjawisk towarzyszących percepcji obrazów oraz ogólnie przyjętych konwencji stosowanych w danej dziedzinie.

## 1.1. Wizualizacja danych

Wizualizacja to techniki umożliwiające tworzenie komunikatów w postaci przekazu wizualnego: rysunków, schematów, diagramów, animacji itp. Techniki te były stosowane od zarania dziejów, o czym świadczą np. prehistoryczne rysunki naskalne, starożytne mapy czy też hieroglify stanowiące zapis dziejów. Obecnie wizualizacja jest bardzo szeroko stosowana w nauce (ang. *scientific visualization*), np. do prezentacji danych pomiarowych (np. medycznych) oraz wyników symulacji (np. chemicznych, materiałowych, astrofizycznych czy też dynamiki płynów), edukacji (ang. *educational visualization*) czy też prezentacji dużych zbiorów abstrakcyjnych informacji (ang. *information visualization*), takich jak np. księgozbiory czy bazy danych. Również w życiu codziennym stykamy się z wizualizacją w postaci *komunikatów wizualnych* (ang. *visual communication*): tekstu (będącego *de facto* zbiorem kształtów), znaków i oznaczeń (np. drogowych), plakatów, fotografii, grafiki itp. Wszystkie te przekazy wizualne niosą ze sobą jakieś informacje. Powszechne używanie komputerów oraz dostęp do uniwersalnych i specjalistycznych programów, umożliwiło zwiększenie możliwości *przekazywania wiedzy* uzupełnianej o przekaz wizualny (np. tabele, szkice, schematy, diagramy) między użytkownikami komputerów (ang. *knowledge visualization*). Jak wi-

dać, rozwój komputerów ułatwił powszechne tworzenie „komunikatów wizualnych", czyli wizualizację na masową skalę.

Głównym zadaniem wizualizacji danych (ang. *data visualization*) zwanej też wizualizacją naukową (ang. *scientific visualization*), jest ułatwienie interaktywnego przeglądania, analizy, prezentacji oraz percepcji abstrakcyjnych danych prezentowanych w postaci graficznej. Rysunek 1.1 przedstawia poszczególne etapy wizualizacji danych wraz z zaznaczonym przepływem informacji między nimi oraz między użytkownikiem.




**Rys. 1.1.** Przepływ informacji w trakcie wizualizacji danych

Źródłem danych jest *Obiekt*, mogący reprezentować zarówno model matematyczny, jak i fizyczny obiekt poddany pomiarom. Do *Obiektu* można przypisać pewien *model koncepcyjny*, będący wyobrażeniem użytkownika o tym, w jaki sposób *Obiekt* działa, czego można się spodziewać w wyniku przeprowadzonego eksperymentu itp. Model koncepcyjny może podlegać modyfikacjom np. pod wpływem analizy wyników pomiarów przedstawianych na wizualizacji. Użytkownik analizując wyniki wizualizacji i porównując je z modelem koncepcyjnym, może dokonywać zmian w procesie wizualizacji w celu jeszcze lepszego zrozumienia zjawisk, jakie zachodzą w *Obiekcie.*

Pierwszym krokiem podczas wizualizacji jest uzyskanie dostępu do danych. Może to być dostęp np. do danych pomiarowych, wcześniej zapisanych w postaci plików lub też wygenerowanych w wyniku przeprowadzonych symulacji. Dane pochodzące z różnych źródeł są następnie poddawane kolejnym transformacjom, podczas których odbywa się główne przetwarzanie mające na celu „wydobycie" istotnych informacji. Transformacje mogą mieć charakter prostego przetwarzania danych (np. skalowanie), jak i bardziej zaawansowanych algorytmów np. filtracji, klasyfikacji czy też segmentacji danych. W trakcie przetwarzania danych wykorzystuje się *model danych,* dobrany do rodzaju danych oraz sposobu ich przetwarzania. Model danych powinien uwzględniać wiedzę dostępną w *modelu koncepcyjnym.*

Wyniki przetwarzania danych są następnie *mapowane* na obiekty graficzne, które z kolei są prezentowane w postaci grafiki np. na ekranie monitora. Może się również zdarzyć, że wyniki prezentacji stają się danymi wejściowymi dla kolejnych transformacji.

Użytkownik analizując otrzymany obraz, może zmienić nastawy algorytmów przetwarzających dane, sposób mapowania lub prezentacji danych. W przypadku zmian dokonywanych w algorytmach przetwarzających dane, mówimy o *eksploracji danych* mającej na celu wydobycie informacji. Zmiany sposobu prezentacji można nazwać *manipulacją danymi*, która ma zazwyczaj na celu polepszenie percepcji. Przykładem może być interaktywna trójwymiarowa wizualizacja przestrzenna, dzięki której można zmieniać pozycję, z której obserwuje się dane, a dzięki temu możliwe jest lepsze zrozumienie tego, na co patrzymy. Wyniki wizualizacji mogą być zapisane w postaci grafiki, animacji lub w innej formie, np. raportu łączącego i podsumowującego uzyskane informacje. Przedstawiony powyżej scenariusz wizualizacji jest jednym z prostszych, ale zawiera wszystkie niezbędne etapy.

## 1.2. Percepcja obrazu przez człowieka

Sposób postrzegania oraz odbioru obrazu przez człowieka jest zagadnieniem bardzo złożonym [War04]. Biorą w nim udział wyspecjalizowane organy, jakimi są oczy oraz rozległe obszary mózgu. Jak wynika z przeprowadzonych badań, zmysł wzroku dostarcza do mózgu największej liczby bodźców [Val05]. Spośród wszystkich połączeń nerwowych łączących receptory zmysłów z mózgiem, ponad 60% stanowią połączenia z oczami. Szacuje się, że dla każdego oka jest to od 1 do 2 milionów połączeń. Dla porównania, drugi co do liczby połączeń zmysł słuchu ma ponad 30 tysięcy połączeń nerwowych. Połączenia te pobudzają w korze mózgowej ponad 500 milionów neuronów w przypadku zmysłu wzroku i 0,8 miliona neuronów w przypadku słuchu. Przytoczone liczby pozwalają oszacować, jak bardzo skomplikowanym *systemem pomiarowym* jest zmysł wzroku. Przetwarzanie przez mózg zawartych w widzianym obrazie informacji wiąże się z gwałtowną redukcją ilości tych informacji. Oszacowano, że w ciągu jednej sekundy do centralnego układu nerwowego dociera około $10^9$ bitów informacji, z czego około 80% przypada na zmysł wzroku. W wyniku przetwarzania informacji, podczas procesu percepcji, do świadomości przedostaje się jedynie ok. $10^2$ bitów [Zab94]. Tak mały strumień informacji wystarcza do podejmowania decyzji i sterowania zachowaniem człowieka.

Na podstawie przeprowadzonych badań neurologicznych, przetwarzanie informacji wizualnej przez człowieka można podzielić na trzy etapy [War04]:

1. **Masowe, równoległe wyodrębnianie niskopoziomowych właściwości i obrazu** rejestrowanego przez fotoreceptory wzroku. Przetwarzanie to odbywa się w neuronach znajdujących się w siatkówce oka oraz przez neurony kory wzrokowej w tylnej części mózgu. Wyspecjalizowane neurony odpowiedzialne są za selektywne wykrywanie określonych (tzw. niskopoziomowych) rodzajów informacji, takich jak: orientacja krawędzi, kolor, faktura czy ruch. Wyspecjalizowane, równoległe przetwarzanie informacji przez biliony neuronów jest bardzo wydajne i jednocześnie niezależne od świadomości na co patrzymy. Podczas tego etapu uzyskiwane informacje mają charakter tymczasowy.

2. **Rozpoznawanie wzorców** odbywa się poprzez podział pola widzenia na regiony i proste modele (np. ciągłe kontury, regiony tego samego koloru lub o podobnej strukturze). W tym etapie wykrywane są również wzorce ruchu. Etap przetwarzania wzorców wykorzystuje ogromne ilości danych, dostarczanych przez pierwszy etap. Dodatkowo korzysta on z pamięci roboczej i długotrwałej oraz uwzględnia świadomość tego, na co patrzymy. Przetwarzanie jest wolne i ma charakter szeregowego przetwarzania informacji. Na tym etapie możliwe jest wykrywanie dowolnych aspektów obrazu. Wynik przetwarzania nazywany jest strumieniem protoobiektów.
3. **Sekwencyjne ukierunkowane przetwarzanie ograniczonej liczby obiektów**. Na tym etapie tylko kilka obiektów (od trzech do pięciu) przechowywanych w pamięci wzrokowej może być analizowanych w jednym czasie. Obiekty te są budowane z dostępnych *modeli* na podstawie tzw. *zapytania wizualnego*. Na przykład, jeśli chcemy użyć mapy drogowej do poszukiwania trasy, wizualne zapytanie spowoduje odszukanie dwóch wizualnych symboli (reprezentujących miasta) połączonych konturem (model drogi).

   Tylko wyniki ostatniego etapu przetwarzania informacji wizyjnej trafiają do innych podsystemów mózgu, np. do systemu werbalnego, w którym do widzianych obiektów zostają przypisane słowa, lub do systemu motorycznego, który spowoduje reakcję ruchu. Wynik przetwarzania stanowi podstawę wizualnego myślenia.

Z zaprezentowanych wyników badań wynika, że sposób przetwarzania obrazu jest skomplikowany, oraz że nie wystarczy „patrzeć" by „widzieć". To, że np. obraz znajomej osoby oraz otaczającego ją tłumu pojawi się na siatkówce oka nie przesądza, że osobę tą zauważymy. By tak się stało, konieczne jest zadziałanie świadomości, czyli pojawienie się zapytania (świadome szukanie konkretnej osoby na podstawie zapamiętanego wzorca lub opisu) lub też muszą zadziałać mechanizmy na pierwszym poziomie przetwarzania obrazu. Może to być wykrycie ruchu wywołanego przez machanie ręką lub też np. wykrycie kontrastującego z otoczeniem koloru ubrania.

### 1.2.1. Zmysł wzroku

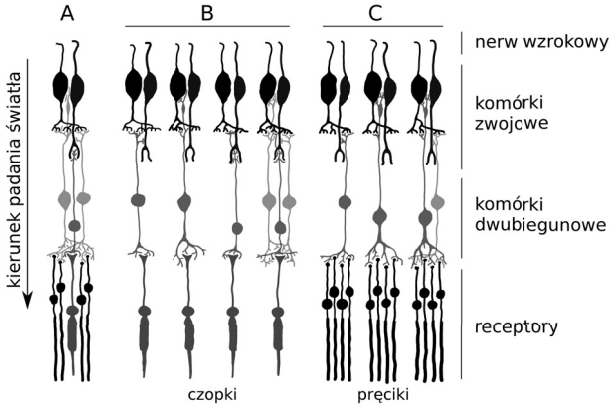
Głównym zadaniem oczu jest odbiór promieniowania świetlnego, a następnie zamiana tego promieniowania na sygnały elektryczne, wstępne przetworzenie i przesłanie sygnałów nerwowych do mózgu, gdzie podlegają dalszemu przetworzeniu. Do głównych zadań zmysłu wzroku należą [Val05]:

**Obrazowanie** – są to procesy zachodzące w oku, których głównym celem jest dostarczenie (poprzez układ optyczny oka) światła do siatkówki. Oko można rozpatrywać jako skupiający układ optyczny, który wytwarza na siatkówce pomniejszony i odwrócony, rzeczywisty obraz obserwowanej przestrzeni.

**Detekcja i rozróżnianie** – na siatkówce dokonywana jest zamiana energii docierającego promieniowania na sygnały elektryczne. Biorą w tym udział komórki receptorów zakończone czopkami oraz pręcikami. W komórkach tych pod wpływem bodźców świetlnych zachodzi odwracalny efekt fotochemiczny. Czopki, odpowiadające za postrzeganie barw, mają małą czułość (działają głównie w dzień, przy dobrym oświetleniu) i jest

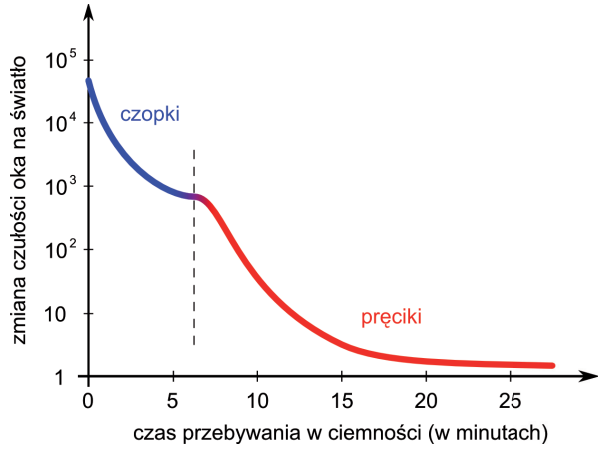
ich około 9 milionów w jednym oku. Pręciki odbierają bardzo małe sygnały (umożliwiają widzenie w ciemnościach), reagują na szerokie pasmo częstotliwości światła i jest ich ponad 100 milionów w jednym oku. Rozkład receptorów na siatkówce nie jest równomierny. Największa ich gęstość jest w centrum siatkówki, w tzw. plamce żółtej o średnicy 2–3 mm. Znajdują się tam jedynie czopki. Wraz ze wzrostem odległości od plamki żółtej, liczba czopków i pręcików maleje i jednocześnie stają się one coraz większe. Siatkówka nie jest jedynie pasywną matrycą rejestrującą obraz. Oprócz fotoreceptorów posiada trzy warstwy sieci wyspecjalizowanych komórek, dzięki którym możliwe jest kluczowe do wykrywania zmian w przestrzeni i czasie wzmacnianie kontrastów i krawędzi.

**Kodowanie oraz transmisja sygnałów** – ponieważ na ponad 100 milionów detektorów przypada jedynie 1–2 miliony połączeń do mózgu (tzw. pasmo wzrokowe zbudowane z włókien nerwowych), fotoreceptory obrazu (pręciki i czopki) łączą się w grupy za pomocą wyspecjalizowanych komórek (dwubiegunowe, horyzontalne, zwojowe oraz amakrynowe). Zadaniem tych grup jest organizacja sygnałów pochodzących z siatkówki, ich dekompozycja, kodowanie oraz przesłanie w głąb mózgu. Na rysunku 1.2 przedstawiono trzy rodzaje grup połączeń, jakie tworzą fotoreceptory. Czopki mają bezpośrednie połączenia (B) z nerwem wzrokowym i pomimo małej czułości umożliwiają postrzeganie szczegółów. Pojedyncze czopki łączą się również w grupy z kilkoma pręcikami (A). Połączone razem grupy do 100 pręcików (C) są najbardziej czułe nawet na najmniejsze zmiany oświetlenia, dzięki efektowi sumowania się stymulacji. Niestety odbywa się to kosztem rozdzielczości przestrzennej. Na rysunku 1.2 nie przedstawiono wszystkich występujących połączeń, np. połączeń receptorów z komórkami horyzontalnymi oraz równoległych połączeń komórek zwojowych poprzez komórki amakrynowe. Obydwa typy komórek umożliwiają łączenie rozległych grup fotoreceptorów, umożliwiając wzajemne oddziaływanie np. hamujące oraz wzmacniające.




**Rys. 1.2.** Podstawowe typy połączeń fotoreceptorów do włókien nerwowych: A – czopki i pręciki połączone razem, B – pojedyncze czopki bezpośrednio podłączone do włókna nerwowego, C – wysokoczuła grupa pręcików

**Adaptacja** – oczy potrafią automatycznie dostrajać się do docierającego do nich strumienia światła. Występują tu dwa zjawiska: szybka zmiana wielkości źrenicy, działającej jak przesłona oraz znacznie wolniejsza adaptacja fotochemiczna siatkówki. Przy zwiększaniu intensywności oświetlenia następuje przełączenie z widzenia nocnego (skotopowego) wykorzystującego jedynie pręciki, przez widzenie zmierzchowe (mezopowe), w którym poza pręcikami zaczynają brać udział czopki, do widzenia dziennego (fotopowego), w którym biorą udział jedynie czopki. Widzenie fotopowe w pełni wykorzystuje możliwości zmysłu wzroku umożliwia postrzeganie barw przy jednoczesnej największej ostrość widzenia. Zmiany czułości receptorów wymagają pewnego czasu, który jest niezbędny do odtworzenia pigmentów. Na rysunku 1.3 przedstawiono zależność progu czułości oka na światło w funkcji czasu. Można zauważyć, że występują dwa obszary: obszar regeneracji czopków oraz pręcików. Ludzkie oko pokrywa bardzo szeroki zakres intensywności, dzięki temu widzimy zarówno w świetle księżyca, jak i w bardzo jasnym świetle słonecznym.




**Rys. 1.3.** Adapatacja fotochemiczna oka – zmiana progu wrażliwości w funkcji czasu, przy braku pobudzeń

**Rozróżnianie** – zarówno w siatkówce, jak i w korze mózgowej występują wyspecjalizowane komórki odpowiedzialne za wstępne przetwarzanie obrazów. Wyróżniono w korze mózgowej kilkadziesiąt ośrodków, które odpowiadają np. za rozróżnianie kierunku zmian w obrazie.

**Identyfikacja, rozpoznawanie i interpretacja** – są to procesy zachodzące w centralnym ośrodku nerwowym (pole wzrokowe, płat potyliczny) z wykorzystaniem wyższych ośrodków, takich jak pamięć czy aktualny kontekst.

Istnieją dwa, w znacznej mierze rozdzielone i zaczynające się już w oczach, szlaki przetwarzania informacji wzrokowej. *Szlak wielkokomórkowy* charakteryzuje się małą rozdzielczością przestrzenną, dużą wrażliwością na kontrast, szybkim przesyłaniem sygnałów bez

informacji o kolorze. W tym szlaku odbywa się analiza widzenia ruchu, lokalizacja w polu widzenia, orientacja przestrzenna oraz postrzeganie głębi i ruchu. *Szlak drobnokomórkowy* ma dużą rozdzielczość przestrzenną, wolniej przesyła informacje w tym o kolorze oraz ma małą wrażliwość na kontrast. W tym torze odbywa się analiza orientacji linii, kształtu, reakcja na kolor oraz rozpoznawanie obiektów.

Odbiór (percepcja) kontrastu, konturów, koloru, tekstury, przestrzeni trójwymiarowej, ruchu i orientacji czy też rozróżnianie obiektów zależy od wzajemnych połączeń i współpracy między wieloma obszarami mózgu. Wszystkie one wpływają na to, w jaki sposób postrzegany i rozumiany jest przekaz wizyjny.

## 1.2.2. Rozdzielczość przestrzenna wzroku

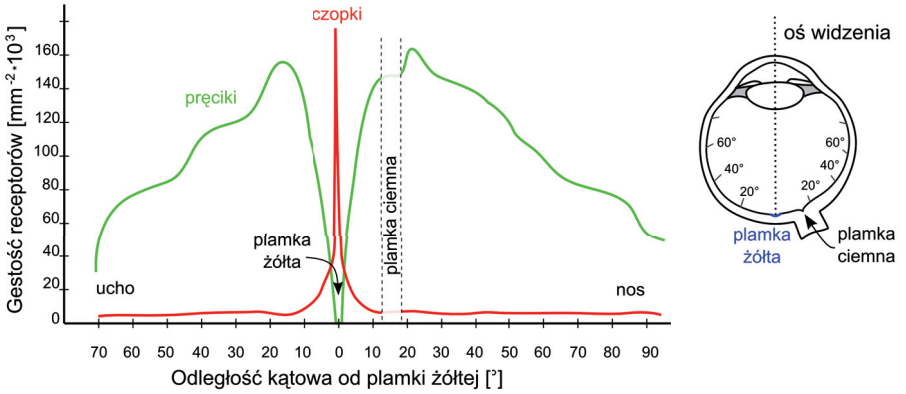
Na rozdzielczość, czyli zdolność do rozróżniania szczegółów w postrzeganym obrazie, wpływają zjawiska optyczne oraz rozkład receptorów na siatkówce [Zab94]. Światło przechodząc przez układ optyczny oka, podlega dyfrakcji na źrenicy oraz aberracji chromatycznej i sferycznej soczewki. Dyfrakcja powoduje, że obraz punktu odtwarzany jest na siatkówce w postaci rozproszonej plamki, zaś punkty świetlne znajdujące się blisko siebie dają obraz w postaci nakładających się plamek. Jeżeli odległość między punktami będzie mniejsza niż połowa średnicy ich plamek, punkty takie nie zostaną zinterpretowane oddzielnie, lecz jako jeden „rozciągnięty" punkt. Warunek ten wynika z tzw. kryterium Rayleigha [Ray]. Aberracje są natomiast spowodowane niejednorodnością współczynnika załamania światła, który jest różny na krawędziach soczewki oka i w jej centrum. Aberracje chromatyczne objawiają się rozszczepieniem światła (tak jak w pryzmacie), co prowadzi do powstawania na krawędziach fałszywych zabarwień. Z kolei aberracja sferyczna objawia się rozmyciem obrazu spowodowanym ogniskowaniem obrazu w różnej odległości od siatkówki.

Dodatkowym czynnikiem ograniczającym rozdzielczość mogą być problemy z akomodacją. Jest to proces, w którym oko dostosowuje się do obserwowanego obrazu przez zmianę geometrii soczewki oka oraz przez wykonywanie mikroruchów. Polegają one na nieustannym świadomym oraz odruchowym zmienianiu kierunku patrzenia, w taki sposób, by obraz padał na najczulszy obszar siatkówki, czyli na plamkę żółtą. Dokonywana jest również zmiana krzywizny soczewki (regulacja ostrości) oraz regulacja ilości wpuszczanego światła (zmiana średnicy źrenicy). Niezdolność mięśni oka do zmiany ogniskowej lub średnicy źrenicy jest przyczyną takich wad wzroku, jak krótkowzroczność, dalekowzroczność lub starczowzroczność. Wady te w znacznym stopniu ograniczają rozdzielczość, jednak mogą być prosto korygowane np. poprzez okulary.

Ze względu na zjawisko dyfrakcji, wymiary źrenicy, długości fali świetlnej oraz wielkość receptorów na siatkówce nawet najmniejszy punkt świetlny zawsze pokrywa kilka czopków w plamce żółtej. Ogranicza to przestrzenną rozdzielczość, z drugiej jednak strony umożliwia zwiększenie wrażliwości oka na intensywność oświetlenia.

Jak wynika z przeprowadzonych doświadczeń, maksymalna rozdzielczość ludzkiego oka wynosi około 1 minutę kątową. Największą rozdzielczość ma obszar siatkówki nazywany plamką żółtą, który obejmuje około 1 stopień kąta pola widzenia. Rysunek 1.4 przedstawia zależność gęstości receptorów w funkcji odległości kątowej od plamki żółtej. Należy zwrócić uwagę, że obszar plamki żółtej zajęty jest wyłącznie przez czopki, nie ma

tam pręcików. Czopki posiadają największą rozdzielczość przestrzenną dzięki bezpośrednim połączeniom z korą mózgową i to one właśnie odpowiadają za ostrość widzenia oraz za postrzeganie barw. Wysokoczułe pręciki wychwytują nawet najmniejsze ilości światła, jednak sygnały z wielu pręcików są zbierane i wzmacniane w pojedynczych neuronach, co ogranicza rozdzielczość. Efekt ograniczonej rozdzielczości można zaobserwować przy bardzo słabym oświetleniu, np. w nocy.




**Rys. 1.4.** Gęstość receptorów na siatkówce oka w funkcji kątowej odległości od plamki żółtej, na podstawie [Ost35]

Rozdzielczość gwałtownie spada wraz z odległością od plamki żółtej: zmniejsza się o połowę w odległości 3° kątowych, zaś już w odległości 5° maleje ośmiokrotnie. Dzięki podświadomym mikroruchom gałek ocznych, możliwe jest wykorzystanie niewielkiego obszaru plamki żółtej do szczegółowej analizy większych obszarów obrazu.

W niewielkiej odległości kątowej (15–20° w kierunku nosa) od najczulszego miejsca na siatkówce, znajduje się obszar o podobnych rozmiarach, zupełnie pozbawiony fotoreceptorów, nazywany *plamką ślepą*, plamką Mariotte'a lub ciemną. Jest to miejsce, w którym znajduje się nerw wzrokowy, łączący siatkówkę z mózgiem. Pomimo braku receptorów, mózg nie zauważa „dziury" w obrazie – brakujące informacje są uzyskiwane albo z obrazu drugiego oka, albo są odtwarzane z wcześniejszego obrazu uzyskanego przy patrzeniu w innym kierunku lub też są wypełniane tłem. Uzupełnianie informacji można zauważyć obserwując rysunek 1.5 przedstawiający dwie litery L oraz P.


**Rys. 1.5.** Obraz umożliwiający sprawdzenie występowania plamki ślepej

Eksperyment należy przeprowadzić w następujący sposób: zamykamy jedno oko (np. lewe), a drugim patrzymy prostopadle do płaszczyzny kartki na literę P. Zmieniając odległość do litery, ale ciągle na nią patrząc, znajdujemy taką odległość, przy której litera L znika z pola widzenia. W takim położeniu obraz tej litery pada dokładnie na plamkę ślepą.

Na rozdzielczość przestrzenną wzroku ma wpływ ilość światła, jaka wpada do oka. Gdy światła jest bardzo mało, sygnały odbierają jedynie pręciki i nie obserwuje się zwiększonej rozdzielczości w plamce żółtej. W miarę zwiększania oświetlenia, zaczynają reagować na bodźce również czopki, najpierw krótko- i średniofalowe, potem również długofalowe (czułe na barwę czerwoną). W takim przypadku rozdzielczość jest największa.

## 1.2.3. Postrzeganie barw

W języku polskim istnieją dwa wyrazy: *kolor* i *barwa*, które w języku potocznym stosowane są wymiennie. Definicja barwy, zaczerpnięta z Encyklopedii PWN, rozróżnia te terminy:

> **barwa**, wrażenie psychofizjologiczne wywołane falami świetlnymi o długości 400–760 nm, a odczuwane za pomocą zmysłu wzroku; barwę charakteryzują: *odcień* (ton, chromatyczność, kolor; nadaje barwie jej nazwę), *natężenie* (jasność), *nasycenie* (maleje wraz z domieszką światła białego) i *czystość* (maleje wraz ze zbliżaniem się do czerni, np. barwa czerwona, czerwonobrązowa, czerń); barwy podstawowe: czerwona (700 nm), zielona (546,1 nm), niebieska (435,8 nm).

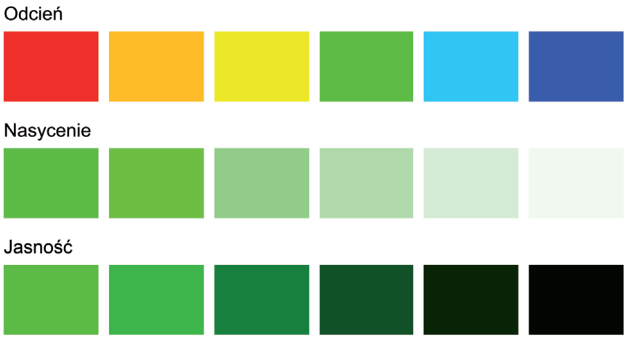
Jak widać, opisanie wrażenia, jakie powstaje podczas patrzenia na „kolorowy świat" nie jest proste. W dodatku wrażenie to jest zupełnie subiektywne i może zależeć od: predyspozycji osobniczych, ewentualnych wad w budowie oka lub ośrodków mózgu związanych ze wzrokiem, oświetlenia, nastroju osoby czy też „wytrenowania" zmysłu wzroku. Zauważono, że osoby zawodowo związane z rozróżnianiem barw zazwyczaj są w stanie rozróżnić więcej odcieni.

W powstającym wrażeniu barwy można wyróżnić cechę *ilościową* zwaną *jasnością* lub *luminancją* (ang. *luminance*), oznaczaną symbolem Y. Jest to moc promieniowania wychodzącego z danej powierzchni w danym kierunku, wyrażona w jednostkach uwzględniających czułość oka dla różnych długości fali [cd/m$^2$]. Jasność jest funkcją nieliniową, zależną od natężenia oświetlenia $E$ [lm] oraz zaadaptowania oka. *Jaskrawość* (ang. *brightness*), oznaczenie B, jest miarą wrażenia wzrokowego, mówiącą czy powierzchnia emituje więcej lub mniej światła. Zdefiniowano również pojęcie jasność dla kolorów, nazywaną *lightness* (*L), która uwzględnia wrażenia percepcyjne (luminancję otoczenia). Poza cechą ilościową wyróżnia się również dwie cechy *jakościowe* określające barwę: *nasycenie* (ang. *saturation*) oraz *odcień* (ang. *hue*). Na rysunku 1.6 przedstawiono podstawowe cechy barwy.

Zmniejszając nasycenie barwy, niezależnie od długości fali, uzyskuje się barwę białą. Zmniejszenie jasności powoduje „dążenie" barwy do koloru czarnego. Czerń, biel oraz wszystkie barwy pośrednie (szarości) nazywane są barwami achromatycznymi.

Wrażenie barwne powstaje na podstawie obserwacji światła, czyli promieniowania elektromagnetycznego, którego długość $\lambda$ mieści się w granicach od 380 do 780 nm. Czułość oka na krańcach tego zakresu (380–400 nm i 700–780 nm) jest bardzo mała, dodatkowo górny przedział długości fal (powyżej 700 nm) nie jest rozróżniany i postrzegany jest jako barwa czerwona. *Barwy proste* inaczej *monochromatyczne* lub *widmowe,* takie jak: czerwona,

zielona czy niebieska, to promieniowanie o bardzo wąskim i ściśle określonym paśmie długości fal. Można je zaobserwować po rozszczepieniu światła białego na pryzmacie. Barwy proste są barwami w pełni nasyconymi. Większość promieniowania docierającego do oka stanowi mieszaninę fal o różnych długościach.



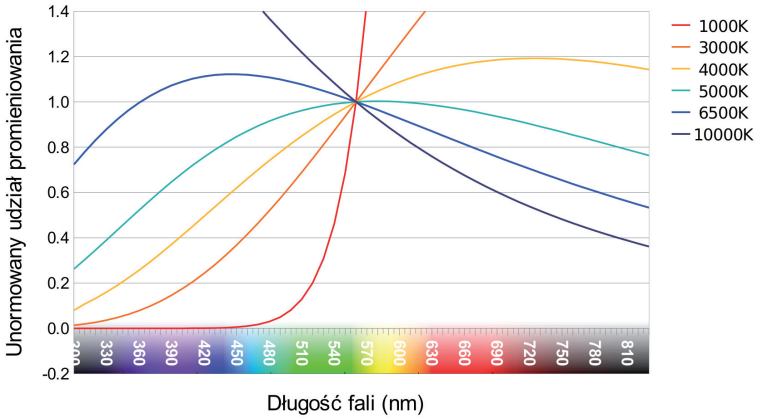

**Rys. 1.6.** Podstawowe cechy barwy: odcień, nasycenie oraz jasność (czystość)

Z każdą długością fali promieniowania docierającego do oka związana jest pewna ilość mocy, można więc wyznaczyć gęstość widmowego rozkładu energetycznego $\varphi(\lambda)$ [W/nm]. Źródła światła, takie jak podgrzane obiekty (np. żarówki wolframowe), emitują zazwyczaj promieniowanie o rozkładzie ciągłym, którego skład zależy od temperatury. Wraz ze wzrostem temperatury zmienia się skład widma emitowanego przez podgrzane ciało promieniowania, zgodnie z prawem Wiena: *wraz ze wzrostem temperatury, maksimum natężenia emitowanego światła przesuwa się w stronę krótszych fal*. Rysunek 1.7 przedstawia udział poszczególnych długości promieniowania w widmie modelu źródła promieniowania ciągłego, jakim jest tzw. *ciało doskonale czarne*, przy zwiększaniu jego temperatury. Na rysunku poszczególne charakterystyki unormowano dla $\lambda = 560$ nm. Dla temperatur poniżej 6000 K promieniowanie postrzegane jest jako ciepłe, czerwone, ponieważ przeważa długofalowe promieniowanie, zaś powyżej temperatury 7000 K jako zimne, o odcieniu niebieskim ze względu na rosnący udział promieniowania o małej długości. Światło białe otrzymuje się, gdy temperatura żarzącego się ciała wynosi ponad 6000 K. Jeżeli rozkład jest stały $\varphi(\lambda) = \text{const}$, wówczas mówimy o świetle białym równoenergetycznym.

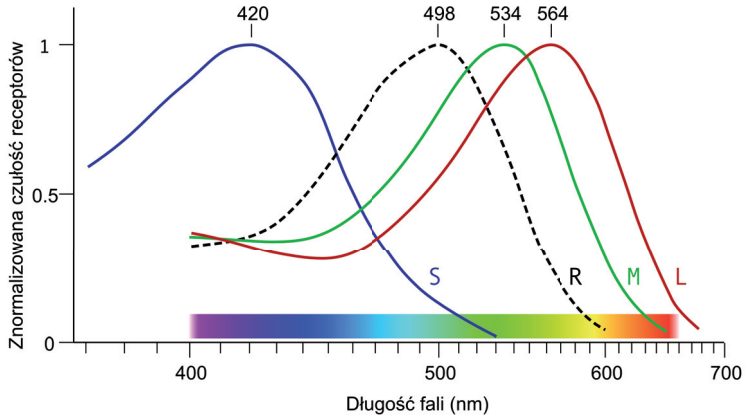
Światło, które sprawia wrażenie białego, zawiera wszystkie długości fal promieniowania widzialnego w odpowiednich proporcjach. Jeżeli w widmie światła można wyróżnić pewien zakres długości fal, dla którego moc promieniowania jest większa niż w pozostałych długościach, wówczas odbierane światło będzie wywoływało wrażenie barwne. Odcień postrzeganej barwy zależy od długości dominujących fal. Nasycenie barwy jest największe, gdy występuje tylko jedna długość fali.

O wrażeniu *jasności* decyduje wielkość strumienia światła przechodzącego przez źrenicę. Strumień światła jest proporcjonalny do natężenia światła (światłości) źródła punktowego lub luminancji powierzchni. Oko posiada różną czułość receptorów dla różnych długości fal promieniowania. Na rysunku 1.8 przedstawiono unormowaną czułość poszczególnych

receptorów w funkcji długości fali promieniowania. Trzy rodzaje czopków uczulone są na trzy różne zakresy promieniowania. Można zauważyć, że podział zakresu widzialnego widma nie jest równomierny.




**Rys. 1.7.** Zależność składu widma φ(λ) od temperatury, unormowana dla λ = 560 nm




**Rys. 1.8.** Znormalizowana czułość trzech rodzajów czopków krótkofalowe (S), średniofalowe (M), długofalowe (L) oraz pręcików (R) w funkcji długości fali

Na najkrótsze długości fal reagują tzw. *czopki krótkofalowe* oznaczone na wykresie literą **S** (ang. *short wavelengths*). Rozmieszczone są na siatkówce w sposób zbliżony do regularnego i jest ich najmniej, około 7% wszystkich czopków na siatkówce ludzkiego oka. Nie występują w centrum tzw. żółtej plamki. Maksimum czułości dla tych receptorów wynosi 420 nm i obejmuje zakres częstotliwości w okolicach barwy niebieskiej, dlatego też można

się spotkać z określeniem „czopki niebieskie". Zakres średnich długości promieniowania obejmują czopki typu **M** (ang. *medium wavelengths*), *średniofalowe* lub „czopki zielone". Dla nich, maksymalna czułość występuje dla λ = 534 nm. Natomiast *czopki długofalowe* **L** (ang. *long wavelengths*) pokrywa zakres promieniowania o największej długości i nazywane są „czopkami czerwonymi". Mają one maksimum czułości dla λ = 564nm. Stosunek liczby czopków L do M wynosi około 1,5. Obydwa rodzaje czopków rozmieszczone są na siatkówce w sposób przypadkowy. Warto zauważyć, że maksima czułości czopków L oraz M znajdują się w niewielkiej odległości.

*Należy zwrócić szczególną uwagę na fakt, że maksymalne czułości czopków* ***nie pokrywają się*** *z tzw. barwami podstawowymi jakimi są czerwień, zieleń oraz kolor niebieski!*

Również pręciki, które służą głównie do widzenia „w ciemnościach", mają czułość zależną od długości obserwowanej fali świetlnej. Czułość pręcików pokrywa środkowy zakres widma, między czopkami S i M. Maksimum występuje dla 498 nm.

**Teoria kolorów przeciwstawnych**

Istnienie trzech rodzajów czopków stanowi podstawę trójbarwnego widzenia kolorów oraz wyjaśnia dlaczego każdą barwę można przedstawić jako sumę dokładnie trzech różnych barw podstawowych. Zauważono jednak, że człowiek postrzega cztery „czyste kolory": czerwony, zielony, żółty oraz niebieski. Są to dwie pary specyficznych kolorów, które nie występują parami jednocześnie: kolor jest albo czerwony, albo zielony i nigdy nie jest postrzegany jako czerwono-zielony.

Twórcą teorii *kolorów przeciwstawnych* (ang. *oponent color theory*) zaprezentowanej w 1892 roku i wyjaśniającej te zjawiska był Ewald Hering. Teoria ta zakłada, że system wzrokowy interpretuje sygnały z czopków w sposób różnicowy. Czopki, reagując na trzy długości fali, dostarczają sygnały tym silniejsze, im długość pobudzającej fali jest bliższa wartości maksymalnej czułości. Tak więc czopki „zawsze" generują odpowiedź, z tym że odpowiedź na pobudzenie pojedynczego fotonu „w maksimum czułości" jest porównywalna z pobudzeniem 10 000 fotonów na skraju czułości. Analiza sygnału różnicowego wydaje się być więc bardziej skuteczna.

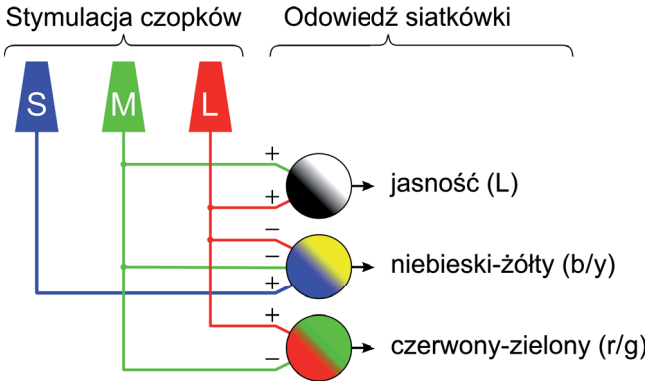
Przeprowadzone badania neurologicznej budowy siatkówki potwierdziły założenia teorii kolorów przeciwstawnych. Sygnały z receptorów koloru nie trafiają bezpośrednio do mózgu, lecz są one wcześniej przetwarzane przez komórki nerwowe siatkówki na trzy tzw. *kanały przeciwstawne.* Rysunek 1.9 przedstawia sposób powstawania kanałów przeciwstawnych z sygnałów będących odpowiedzią czopków na stymulację światłem.

Komórki zwojowe tworzą na siatkówce koncentryczne struktury tzw. *pola percepcyjne* o dwóch obszarach: centralnym oraz je otaczającym. Obszar centralny może reagować na pobudzenie światła poprzez zwiększenie (ang. *ON-center*) lub zmniejszenie (ang. *OFF-center*) swojej aktywności.

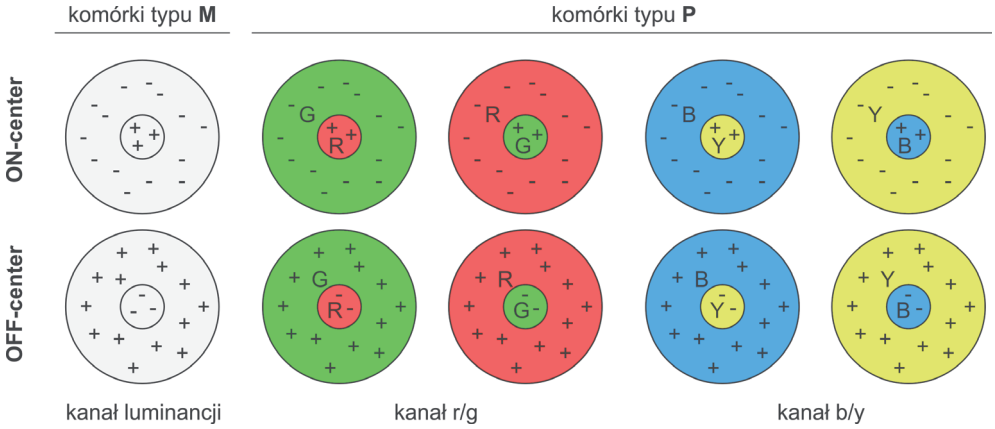
Obszar otaczający centrum zawsze reaguje w sposób przeciwny do centrum. Spośród takich połączeń komórek zwojowych wyróżniono dwa rodzaje: komórki typu M oraz typu P. Komórki typu M stanowią 8% wszystkich występujących na siatkówce komórek zwojowych. Charakteryzują się tym, że mają małe wymiary, są połączone z pręcikami i czopkami, a ich odpowiedź nie zależy od barwy, a jedynie od jasności. To one odpowiadają za tworzenie sygnałów

w kanale luminancji. Komórki typu P są znacznie liczniejsze (ok. 80%), mają znacznie większe wymiary, łączą przede wszystkim czopki i tworzą obydwa przeciwstawne kanały koloru.

Rysunek 1.10 prezentuje występujące w siatkówce pola obrazowe. Znakiem plusa zaznaczono obszary reagujące wzrostem aktywności na pobudzenie, zaś minusem te, które działają hamująco na pobudzenie zmniejszając swoją odpowiedź przy wzroście pobudzenia.

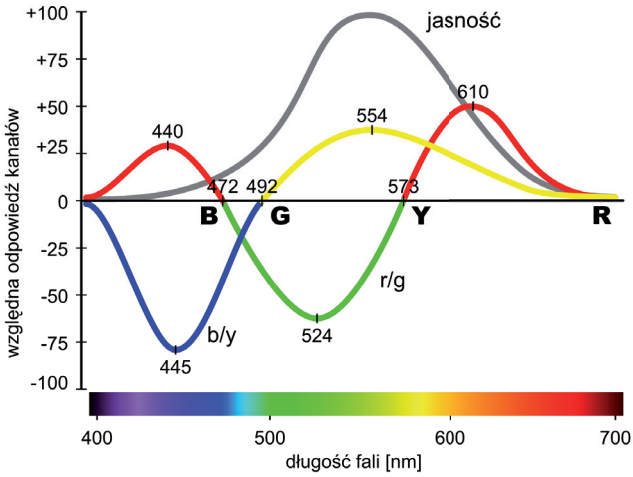



**Rys. 1.9.** Odpowiedź siatkówki na pobudzenie światłem czopków odbywa się w trzech kanałach przeciwsobnych




**Rys. 1.10.** Podstawowe rodzaje komórek zwojowych tworzące pola obrazowe, które są źródłem sygnałów w kanałach przeciwstawnych luminancji oraz kolorów r/g i b/y

Sygnał w *kanale luminancji* powstaje z sumy bodźców dostarczanych przez czopki L oraz M. Różnica ich odpowiedzi tworzy *kanał czerwony-zielony* (r/g). *Kanał niebieski-żółty* (b/y) powstaje poprzez odjęcie od sygnału z czopków krótkofalowych S sygnału luminancji.




**Rys. 1.11.** Względna odpowiedź siatkówki dla światła widzialnego: funkcje kanałów przeciwstawnych luminancji, r/g oraz y/b (Hurvich & Jameson (1955))

Na rysunku 1.11 przedstawiono względną odpowiedź siatkówki w poszczególnych kanałach przeciwstawnych luminancji, (r/g) oraz (b/y) na pobudzenie w całym zakresie promieniowania widzialnego. W miejscach przecięcia krzywych z osią odciętych zaznaczono współrzędne „czystych kolorów": niebieskiego (punkt B), zielonego (G) i żółtego (Y). Kolor czerwony (R) nie jest określony przez jeden punkt i znajduje się na skraju obszaru percepcji. W oznaczonych punktach występuje odpowiedź jedynie w jednym z kanałów koloru.

Można więc powiedzieć, że mózg „nie widzi" wczesnego przetwarzania informacji jakim są „trójbarwne" sygnały z fotodetektorów, a jedynie sygnały różnicowe zakodowane w dwuwymiarowej przestrzeni barw wyznaczanej przez kanały (r/b) i (y/b), uzupełnionej o trzeci wymiar niosący informację o jasności (kanał luminancji).

Światło padając na obiekt rzeczywisty, jest w części odbijane oraz pochłaniane zgodnie z widmowymi, indywidualnymi charakterystykami absorpcji $\tau(\lambda)$ oraz odbicia $\rho(\lambda)$. To, w jaki sposób postrzegamy daną powierzchnię, zależy od właściwości powierzchni, czyli charakterystyk $\tau(\lambda)$ i $\rho(\lambda)$ oraz widma promieniowania, jakie pada na obiekt. Taka sama powierzchnia może powodować różne wrażenia barwne w różnym oświetleniu!

Człowiek ma bardzo szeroką czułość na światło – widzi przy niewielkiej luminancji (od około $10^{-6}$ cd/m$^2$), jak i przy bardzo dużej (do $10^6$ cd/m$^2$). Jest to częściowo możliwe dzięki zmianie średnicy źrenicy oraz przede wszystkim dzięki mechanizmowi *adaptacji wzroku.* Odpowiada on między innymi za subiektywne zmniejszanie jasności części obserwowanego obrazu, przy jednoczesnym pozostawieniu najjaśniejszych fragmentów bez zmian. Adaptacja odpowiada również za wykorzystanie konkretnego typu receptorów. Przy słabym oświetleniu wykorzystywane są pręciki – receptory uczulone jedynie na intensywność promieniowania. Przy silnym oświetleniu wykorzystywane są głównie czopki, które odpowiadają za rozróżnianie kolorów. Dodatkowo, przy bardzo silnym oświetleniu zaobserwowano mecha-

nizmy zmniejszania wymiarów receptorów [Val05] chroniące przed absorpcją zbyt dużej dawki energii promieniowania.

Poza adaptacją na natężenie światła, wyróżniane jest również zjawisko *adaptacji barw*, polegające na zachowaniu stałości subiektywnego wrażenia barwy, niezależnie np. od oświetlenia. Dzięki temu zjawisku możliwe jest rozróżnienie dwóch zbliżonych kolorów niezależnie od oświetlenia. Zjawisko adaptacji nie występuje w fotografii cyfrowej, gdzie stosuje się krzemowe sensory obrazu. Dla poprawnego oddania tak zarejestrowanych kolorów niezbędne jest wprowadzenie kompensacji za pomocą tzw. balansu bieli. Występujący w mózgu mechanizm adaptacji do kolorów zapewnia stałość subiektywnego wrażenia barwy w dużej mierze niezależnie od składu widma światła.

## 1.2.4. Kontrast

Kontrast stanowi podstawę widzenia. Jak wynika z badań przeprowadzonych nad działaniem siatkówki, do mózgu trafiają sygnały względne, będące gradientową odpowiedzią na pobudzenie komórek nerwowych znajdujących się w siatkówce. Występujące w siatkówce neurony (fotoreceptory, horyzontalne, dwubiegunowe i amakrynowe) tworzą połączenia od fotoreceptorów do mózgu (tzw. pionowa droga sygnału), ale również wymieniają sygnały między sobą (tzw. droga pozioma). Dzięki tym połączeniom możliwe jest tworzenie wyspecjalizowanych grup współpracujących ze sobą neuronów. Do ich głównych zadań należy wzmacnianie kontrastów, podkreślanie zmian w przestrzeni i czasie, wzmacnianie krawędzi oraz wykrywanie barwy.

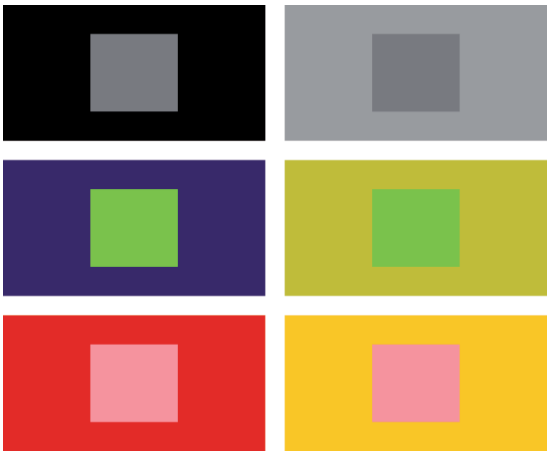
Wzajemny kontrast jest osiągany poprzez tzw. *hamowanie oboczne*, czyli wzajemnie hamujące oddziaływanie między receptorami, kiedy to ciemny obszar „hamuje" jasny obszar sąsiadujący z nim i *vice versa*. Na rysunku 1.12 przedstawiono przykład hamującego działania neuronów na siebie. We wszystkich przypadkach luminancja wewnętrznego kwadratu jest identyczna, jednak na skutek oddziaływania coraz ciemniejszego otoczenia, wewnętrzne kwadraty postrzegane są jako coraz jaśniejsze. Otoczenia wzmacnia efekt różnicy jasności.

**Rys. 1.12.** Wpływ tła na postrzeganie jasności

Również w przypadku postrzegania barw można zauważyć podobne efekty jak przy postrzeganiu luminancji. Przykładem jest rysunek 1.13. Wzmacnianie kontrastu w przypadku barw jest szczególnie wyraźne dla kolorów dopełniających się, czyli wówczas, gdy występuje oddziaływanie między grupami neuronów (patrz początek rozdziału 1.2.1)

Odpowiedź na zmianę kontrastu nie jest liniowa. Własność tą opisuje prawo Webera–Fechnera, łączące relację pomiędzy fizyczną miarą bodźca a reakcją układu biologicznego.

Prawo to głosi: *Wartość reakcji układu biologicznego jest proporcjonalna do logarytmu bodźca.* Jest to prawo fenomenologiczne będące wynikiem wielu obserwacji praktycznych. Wielkość zauważalnego progu kontrastu zależy również od aktualnego stanu adaptacji oka oraz wielkości kątowej obserwowanego obiektu.

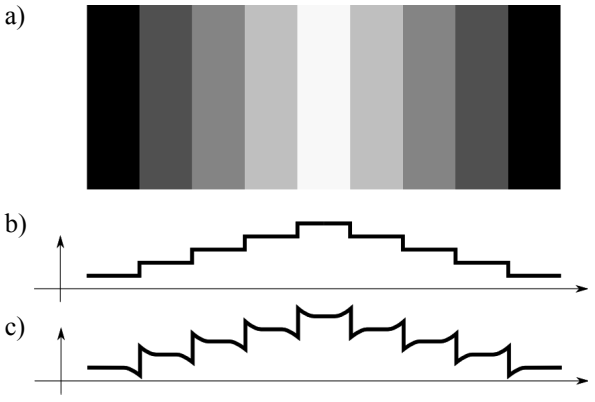


**Rys. 1.13.** Kontrast barwy: środkowe kwadraty, mające taki sam kolor, postrzegane są jednak w różny sposób: na ciemnym tle są jaśniejsze, zaś na jasnym ciemniejsze

Rysunek 1.14 przedstawia diagram Campbell–Robson ukazujący zdolność rozróżniania szczegółów w obrazie, czyli wrażliwość na kontrast w funkcji częstotliwości bodźca. Wartość jasności poszczególnych pikseli modulowana jest sinusoidalnie w osi pionowej, przy czym częstotliwość modulacji wzrasta logarytmicznie. Dodatkowo w osi poziomej zmieniany jest logarytmicznie kontrast od 0,5% do 100%. Zgodnie z przytoczonym prawem Webera–Fechnera, zastosowanie skali logarytmicznej powinno zapewnić proporcjonalną reakcję wzroku. Na wykresie tym można zauważyć, że zdolność do rozróżniania szczegółów silnie zależy od kontrastu. Objawia się to tym, że część obszaru postrzegana jest jako „jednorodna plama" koloru szarego. Dla najmniejszych wartości kontrastu rozróżniane są jedynie wąskie zakresy częstotliwości. Zakres tych częstotliwości zmienia się wraz ze zmianą odległości między okiem a obrazem. Obszar „szarej plamy" zwiększa się przy zwiększaniu odległości. Również przy zbyt małej odległości zmienia się zakres rozróżnianych prążków, szczególnie o małej częstotliwości. Obszar, w którym nie rozróżnia się szczegółów, ma kształt litery C i obejmuje lewą krawędź wraz z górnym oraz dolnym rogiem.

Oko ma właściwości wzmacniania kontrastu. Jest to szczególnie widoczne w postaci *efektu Macha*, który występuje na styku obszarów o różnej i jednocześnie stałej jasności. Styk tych obszarów wydaje się być cieniowany (zmienna jasność). Efekt ten jest wynikiem wzajemnego hamującego oddziaływania sąsiadujących receptorów. Na rysunku 1.15 przedstawiono przykład obszarów o rożnej jasności (a), faktyczną zmianę jasności (b) oraz postrzeganą jasność (c).


**Rys. 1.14.** Zdolność rozróżniania szczegółów zależy od częstotliwości zmian w obrazie oraz kontrastu (diagram Campbell–Robson)



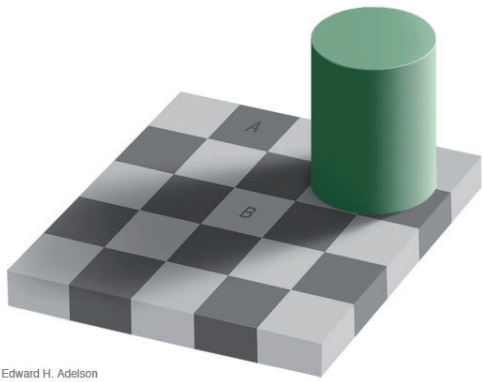

**Rys. 1.15.** Przykład efektu pasm Macha (a), rzeczywista (b) oraz postrzegana (c) jasność

Innym zjawiskiem występującym na granicy obszarów o różnej jasności jest *irradiacja,* czyli zjawisko powodujące pozorne zwiększanie jaśniejszych obszarów, znajdujących się na ciemniejszym tle (rys. 1.16). Spowodowane to jest rozprzestrzenianiem się światła, na skutek dyfrakcji, z jasnego obszaru na sąsiednie receptory siatkówki. Zjawisko to może być szczególnie istotne, gdy ważna jest percepcja rozmiarów.

**Rys. 1.16.** Przykład zjawiska irradiacji, czyli pozornej zmiany wielkości

Przykładem błędnej interpretacji jasności może być rysunek 1.17. na którym obszar A postrzegany jest jako ciemniejszy, zaś obszar B jako jaśniejszy. W rzeczywistości mają identyczny poziom jasności. Można to sprawdzić, zasłaniając obszar między oznaczonymi kwadratami. To otoczenie determinuje, w jaki sposób odbierany jest konkretny obszar.

Biorąc pod uwagę sposób percepcji jasności można stwierdzić, że skala szarości jest bardzo zawodną metodą przekazywania informacji o charakterze ilościowym.




**Rys. 1.17.** Przykład postrzegania kontrastu: obszary A oraz B mają taki sam odcień

## 1.2.5. Bezwładność wzroku

Bezwładność wzroku umożliwia obserwację nieciągłych bodźców jako zjawisk ciągłych. Dzięki temu, że wrażenie wzrokowe utrzymuje się przez krótki czas możemy oglądać filmy w kinie i telewizji. Mechanizm odbioru impulsu światła składa się z kilku etapów. Po

wystąpieniu bodźca następuje okres utajenia, po którym pojawia się wrażenie światła. Długość okresu utajenia waha się w granicach 50...200 ms. Czas ten zależy od intensywności bodźca, długości fali światła, pobudzanego obszaru siatkówki i stanu adaptacji siatkówki. Po okresie utajenia pojawia się obraz pierwotny, który utrzymuje się przez dłuższy czas. Intensywność obrazu pierwotnego na początku gwałtownie narasta do wartości maksymalnej, a następnie równie gwałtownie zmniejsza się do pewnej wartości, która malejąc, utrzymuje się jeszcze przez 50...200 ms. Czas ten zmienia się odwrotnie proporcjonalnie do intensywności bodźca.

Jeżeli częstotliwość powtórzeń bodźców jest większa niż tzw. *częstotliwość krytyczna migotania*, wówczas powstaje wrażenie obrazu ciągłego. Częstotliwość krytyczna migotania głównie zależy od luminancji bodźca oraz innych wielkości, takich jak: wielkość pola migoczącego czy też zdolność adaptacji oka. Przyjmuje się, że dla źródeł światła o dużej luminancji, np. pochodzącego z monitora kineskopowego, częstotliwość odświeżania nie powinna być mniejsza niż 60–80 Hz. W przypadku projekcji stereoskopowej, czyli takiej, która dostarcza niezależne obrazy dla obu oczu, również niezbędne jest zapewnienie wysokiej częstotliwości odświeżania. Przy stosowaniu okularów migawkowych należy szczególnie uwzględnić czas przesłaniania oka, który może przeszkadzać i wzmacniać poczucie nieciągłości obrazu.

## 1.2.6. Ostrość obrazu, postrzeganie przestrzeni

*Głębokość pola* dla układu optycznego oka to obszar, przez który obserwowany przedmiot może być przesuwany w przestrzeni bez wywoływania dostrzegalnego zacierania (rozmycia) się jego obrazu [Zab94]. Głębokość pola jest odpowiednikiem zjawiska *głębi ostrości* znanej z fotografii. Wielkość otworu źrenicy zmienia się podczas akomodacji. Otwór źrenicy określa głębokość pola. I tak dla otworu 2 mm głębokość pola wynosi od nieskończoności do około 15 m, a dla źrenicy 4 mm – od nieskończoności do 30 m. Zwiększanie otworu źrenicy powoduje zmniejszanie pola głębokości. Wszystkie punkty obserwowanej trójwymiarowej sceny znajdujące się poza głębią ostrości obrazowane są na siatkówce w postaci kół, tzw. plamek rozproszenia. Jeżeli rozmiary plamki rozproszenia są większe niż rozmiary receptorów wzroku, obraz staje się nieostry.

Zjawisko rozmywania obrazu jest wykorzystywane podczas percepcji do określania odległości oraz wzajemnego położenia między obiektami w przestrzeni. Patrząc daleko, nie widzimy ostro obiektów najbliższych i na odwrót. Zjawisko to może być wykorzystane do zwrócenia uwagi na konkretne szczegóły, np. w wizualizacji poprzez sztuczne rozmycie części nieistotnych.

Postrzeganie przestrzeni możliwe jest również dzięki zdolności do niezależnej zmiany pozycji gałek ocznych, tzw. konwergencji. Dzięki temu możliwe jest śledzenie obiektów w przestrzeni oraz określanie odległości od przedmiotu. Odbywa się to poprzez analizę różnic w odbieranych przez oczy obrazach (widzenie steroskopowe) z uwzględnieniem informacji o tym, gdzie zostały skierowane przez mięśnie gałki oczne. Możliwe jest wywołanie wrażenia widzenia przestrzennego (stereoskopia) przez dostarczenie do poszczególnych oczu odpowiednio „spreparowanych" obrazów dwuwymiarowych. Mózg na podstawie występujących różnic w obrazach jest w stanie odtworzyć informację o przestrzennych zależnościach.

# 1.3. Modele barw

Opisanie subiektywnego wrażenia, jakim jest barwa, jest sporym wyzwaniem. Na potrzeby technologii wydruku oraz wyświetlania i przetwarzania obrazów powstały różne modele barw. Można je podzielić na następujące grupy:

- opracowane na podstawie percepcji koloru (np. modele opracowane przez Międzynarodową Komisję Oświetlenia CIE [1]),
- modele odwzorowujące proces powstawania koloru w konkretnych technologiach (RGB, CMY czy CMYK),
- modele barwne oparte na systemach specjalnie przygotowanych fizycznych wzorców (np. system PANTONE® posługuje się wzornikami barw z przypisanymi im kodami).

W dalszej części rozdziału zostaną przedstawione informacje na temat podstawowych modeli barw, stosowanych głównie w grafice komputerowej oraz wizualizacji.

## 1.3.1. Modele CIE

Międzynarodowa Komisja Oświetlenia (CIE – Comission Internationale de l'Eclairage) opracowała całą rodzinę modeli opierających się na eksperymentalnych badaniach trójchromatycznego sposobu postrzegania barw przez człowieka. Modele te opisują, *jak barwa wygląda*, a nie *jak jest tworzona* na konkretnym urządzeniu.

Jednym z najczęściej obecnie stosowanych modeli barwnych niezależnych od sprzętu jest, opracowany w 1931 roku, model CIE XYZ. Model ten jest trójwymiarową przestrzenią barw, uwzględniającą właściwości światła oraz percepcję barw przez ludzkie oko. Model CIE XYZ wywodzi się z opracowanego na podstawie pomiarów kolorymetrycznych [2] systemu CIE RGB, zwanego również *przestrzenią bodźców fizycznych.*

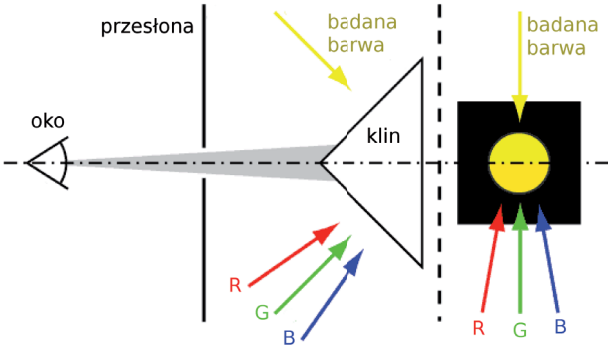
**Model CIE RGB**

W modelu CIE RGB doświadczalnie określono wartości tzw. *składowych trójchromatycznych* $\overline{r}(\lambda)$, $\overline{g}(\lambda)$ oraz $\overline{b}(\lambda)$. Jako punkt wyjścia wykorzystano trójchromatyczną teorię percepcji barw, czyli fakt, że ludzki zmysł wzroku posiada trzy rodzaje receptorów barwy.

Wartości składowych trójchromatycznych wyznaczono dla wszystkich barw widmowych, przy zachowaniu stałości strumienia energetycznego promieniowania, przez porównanie badanej barwy rozszczepionego światła białego ze światłem wzorcowym. Rysunek 1.18 przedstawia zasadę działania kolorymetru. Światło wzorcowe było addytywną mieszaniną trzech długości fal: dwóch prążków występujących w widmie światła łuku rtęciowego o długościach $\lambda_G$ = 546,1 nm i $\lambda_B$ = 435,8 nm oraz górnoprzepustowo przefiltrowanego światła lampy żarowej o $\lambda_R$ > 700 nm. Udział poszczególnych długości fal regulowany był za pomocą przesłony.

[1] Międzynarodowa Komisja Oświetleniowa, franc. Comission Internationale de l'Eclairage, strona domowa `http://cie.co.at`

[2] Kolorymetria: technika pomiarowa, której zadaniem jest ilościowy opis barwy




**Rys. 1.18.** Zasada działania kolorymetru, umożliwiającego wyznaczenie składowych trójchromatyczych badanej barwy poprzez porównanie jej z barwą wzorcową

Barwę badaną ($C$) uznaje się za identyczną z wzorcową, gdy nie można ich wzrokowo rozróżnić. Wielkości otrzymanych nastaw przesłon $\tau_R(\lambda)$, $\tau_G(\lambda)$ i $\tau_B(\lambda)$) normuje się do uzyskanych podczas kalibracji nastaw przesłon dla światła białego $\tau_R(W)$, $\tau_G(W)$, $\tau_B(W)$:

$$\begin{aligned} \overline{r} &= \frac{\tau_R(\lambda)}{\tau_R(W)}, \\ \overline{g} &= \frac{\tau_G(\lambda)}{\tau_G(W)}, \\ \overline{b} &= \frac{\tau_B(\lambda)}{\tau_B(W)}. \end{aligned} \tag{1.1}$$

Dowolną obserwowaną barwę $C$ można więc opisać w postaci tzw. *równania trójchromatycznego*:

$$[C] \equiv \overline{r} * [R] + \overline{g} * [G] + \overline{b} * [B], \tag{1.2}$$

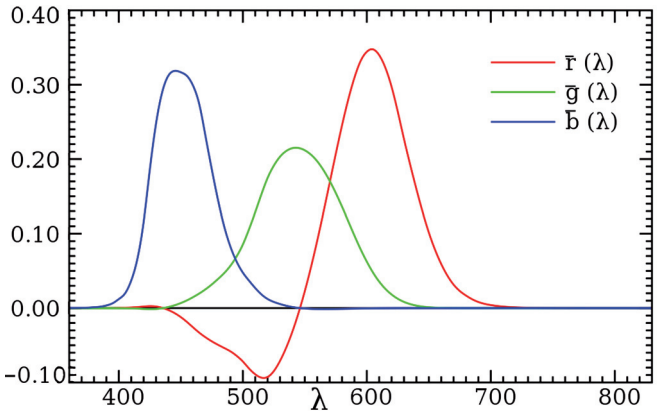
gdzie $[R]$, $[G]$, $[B]$ są wersorami przestrzeni CIE RGB, odpowiadającymi długościom fal pobudzeń $\lambda_R$, $\lambda_G$, $\lambda_B$.

Składowe trójchromatyczne $\overline{r}$, $\overline{g}$ *i* $\overline{b}$ są wielkościami bezwymiarowymi, oznaczeniami trzech podstawowych składowych (czerwonej, zielonej i niebieskiej), biorących udział w powstawaniu wrażenia barwy. Wzorcowe światło białe ma wszystkie składowe równe 1. Na rysunku 1.19 przedstawiono wyznaczone wartości składowych trójchromatycznych w funkcji długości fali dla modelu CIE RGB.

Można zauważyć, że w modelu CIE RGB występują ujemne wartości składowej $\overline{r}$. Nie mają one fizycznej interpretacji, a są jedynie wynikiem sposobu określania składowych względem barwy białej (1.2). Przyjęta baza [R,G,B] umożliwia uzyskanie jedynie barw wynikających z ich addytywnego mieszania. Niestety, dla dowolnie dobranej trójki barw podstawowych zawsze istnieją barwy, których nie można uzyskać jedynie poprzez dodawanie tych barw. Są to np. barwy o większej luminancji niż maksymalna porcja barw podstawowych albo barwy bardziej nasycone od barw podstawowych. Barwy takie można opisać w wybra-

nym układzie trójchromatycznym poprzez osłabienie bodźca (zmniejszenie luminancji np. filtrem szarym) lub zmniejszenie nasycenia przez zmieszanie z barwami podstawowymi. Uwzględniając współczynniki $\overline{r^c}$, $\overline{g^c}$, $\overline{b^c}$ zmniejszające intensywność bodźca, równanie trójchromatyczne przyjmuje postać:

$$[C] \equiv \frac{\overline{r} - \overline{r^c}}{t_c} * [R] + \frac{\overline{g} - \overline{g^c}}{t_c} * [G] + \frac{\overline{b} - \overline{b^c}}{t_c} * [B]. \tag{1.3}$$




**Rys. 1.19.** Składowe trójchromatyczne widmowe modelu CIE RGB

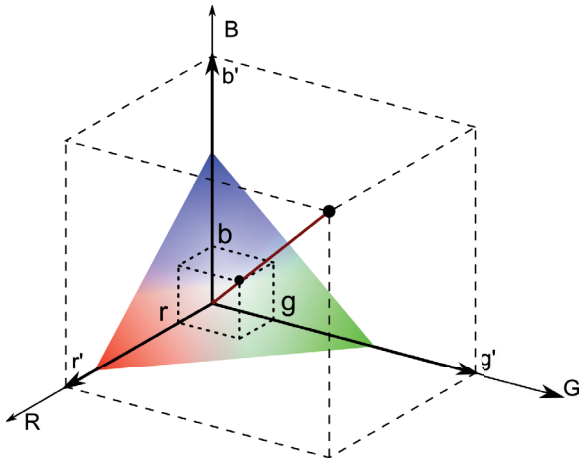
Z równania (1.3) wynika, że każda ze składowych trójchromatycznych może być liczbą ujemną. Występowanie wartości ujemnych stanowi główną wadę przestrzeni CIE RGB, bazującej na bodźcach fizykalnych.

Model CIE RGB rozpięty jest na trzech ortogonalnych wektorach R, G i B, które reprezentują trzy barwy podstawowe, użyte do tworzenia barwy wzorcowej. Wektory te tworzą kartezjański układ współrzędnych, w którym można zaznaczyć składowe trójchromatyczne $\overline{r}$, $\overline{g}$ i $\overline{b}$. Uwzględniając zmierzony udział poszczególnych składowych w równoenergetycznym świetle białym, przez przeskalowanie każdej z osi, otrzymuje się tzw. *jednostki trójchromatyczne.* Przy tak dobranych jednostkach trójchromatycznych luminancja bodźców podstawowych $L_R$, $L_G$, $L_B$ pozostaje w następującym stosunku: $L_R : L_G : L_B = 1 : 4{,}5907 : 0{,}0601$. W tak utworzonym układzie współrzędnych, na osiach odkłada się wartości składowych trójchromatycznych. Odległość konkretnego punktu, opisującego badaną barwę od początku układu współrzędnych, związana jest ze strumieniem światła.

W celu wyznaczenia wartości opisujących wyłącznie barwę, określa się *współrzędne trójchromatyczne*: *r*, *g* oraz *b*. Otrzymuje się je, wyznaczając punkt przecięcia wektora, łączącego początek układu współrzędnych z punktem ($\overline{r}$, $\overline{g}$ i $\overline{b}$) opisującym barwę w układzie RGB, z płaszczyzną przechodzącą przez *jednostki trójchromatyczne.* Płaszczyzna ta opisana jest równaniem: $(\overline{r} + \overline{g} + \overline{b}) = 1$. Wartości współrzędnych trójchromatycznych *r*, *g* i *b* wyznaczane są z następującej zależności (1.4):

$$
\begin{aligned}
r &= \frac{\overline{r}}{\overline{r}+\overline{g}+\overline{b}},\\
g &= \frac{\overline{g}}{\overline{r}+\overline{g}+\overline{b}},\\
b &= \frac{\overline{b}}{\overline{r}+\overline{g}+\overline{b}}.
\end{aligned}
\tag{1.4}
$$

Na rysunku 1.20 przedstawiono interpretację graficzną przestrzeni CIE RGB z zaznaczonymi: składowymi trójchromatycznymi ($\overline{r}$, $\overline{g}$ i $\overline{b}$), współrzędnymi trójchromatycznymi ($r$, $g$ i $b$) oraz zaznaczoną kolorami płaszczyzną rozpiętą na jednostkach trójchromatycznych.




**Rys. 1.20.** Współrzędne trójchromatyczne przestrzeni CIE RGB dla widma równoenergetycznego

**Model CIE XYZ**

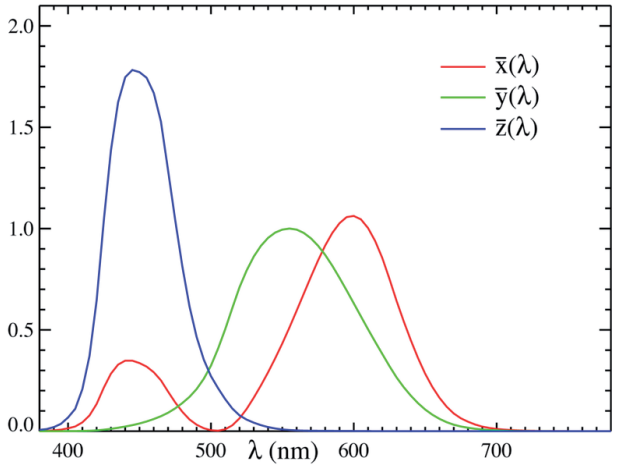
Przestrzeń trójchromatyczna CIE XYZ nie ma wad przestrzeni CIE RGB, czyli: współrzędne trójchromatyczne wszystkich barw psychofizycznych są liczbami nieujemnymi oraz dodatkowo punkt chromatyczności bieli równoenergetycznej $E$ leży blisko środka ciężkości tzw. krzywej barw widmowych. Osiągnięto to poprzez liniową transformację przestrzeni CIE RGB następującym równaniem:

$$
\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \frac{1}{0{,}17697} \begin{bmatrix} 0{,}49 & 0{,}31 & 0{,}20 \\ 0{,}17697 & 0{,}81240 & 0{,}01063 \\ 0{,}0 & 0{,}01 & 0{,}99 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \tag{1.5}
$$

Zastosowana transformacja zmieniła bazę poprzez dobranie innych bodźców podstawowych. Otrzymane trzy nowe „fikcyjne” barwy podstawowe, dla których nie istnieją rzeczywiste pobudzenia, oznaczono literami X, Y oraz Z. Pobudzenia dobrano w taki sposób,

że Y odpowiada jedynie za luminancję (jaskrawość), zaś bodźce X i Z zawierają informację o chromatyczności barwy, a ich luminancja jest zerowa.

Rysunek 1.21 jest odpowiednikiem rysunku 1.19 i przedstawia zależność amplitudy nowych współrzędnych trójchromatycznych $(x,y,z)$ w funkcji długości fali λ, wyznaczonych dla promieniowania równoenergetycznego. Można zauważyć, że wszystkie składowe posiadają jedynie wartości dodatnie w całym zakresie długości fal promieniowania.




**Rys. 1.21.** Współrzędne trójchrornatyczne przestrzeni CIE XYZ dla widma równoenergetycznego

Tabela 1.1 zawiera współrzędne podstawowych kolorów: czerwonego, zielonego, niebieskiego oraz bieli określone we współrzędnych CIE XYZ opracowane dla standardu HDTV (sRGB).

**Tabela 1.1**

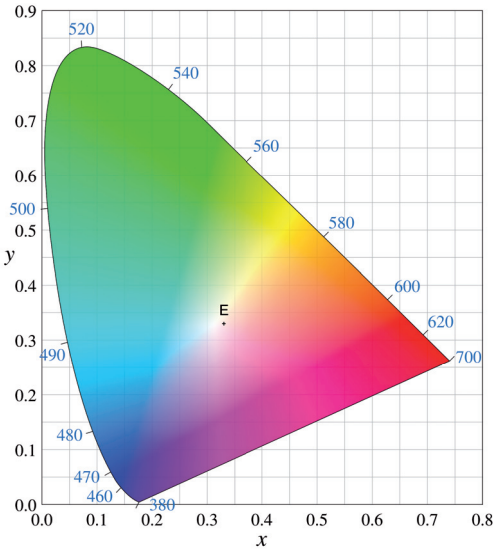
Współrzędne podstawowych kolorów określone dla standardu HDTV w przestrzeni CIE XYZ

| | czerwony (R) | zielony (G) | niebieski (B) | biały |
|---|---|---|---|---|
| *x* | 0,640 | 0,300 | 0,150 | 0,3127 |
| *y* | 0,330 | 0,600 | 0,060 | 0,3290 |
| *z* | 0,030 | 0,100 | 0,790 | 0,3582 |

### Wykres chromatyczności CIE

W praktyce bardzo często korzysta się z tzw. *wykresu chromatyczności* przedstawionego na rysunku 1.22. Na brzegu zaznaczonego obszaru, znajdują się nasycone barwy widmowe wraz z odpowiadającymi im długościami fal. Punkty te (widmowe składowe trójchrornatycz-

ne) tworzą *krzywą barw widmowych*. Na odcinku łączącym punkty reprezentujące skrajne długość widzialnego promieniowania (380 nm i 700 nm) znajdują się *barwy niespektralne* – purpura i magenta. Są to barwy, które nie występują bezpośrednio w widmie, a są jedynie wrażeniem wywoływanym pod wpływem mieszaniny fal o kilku długościach. Kolor biały równoenergetyczny oznaczono punktem *E* i ma on w tym modelu współrzędne $x = y = z = 1/3$.




**Rys. 1.22.** Wykres chrominancji: przestrzeń barw CIEYxy

Wartości współrzędnych trójchromatycznych *x*, *y*, *z* zależnych od pobudzenia fikcyjnych (teoretycznych) receptorów *X*, *Y*, *Z* opisane są następującymi zależnościami:

$$x = \frac{X}{X+Y+Z},$$
$$y = \frac{Y}{X+Y+Z}, \tag{1.6}$$
$$z = \frac{Z}{X+Y+Z}.$$

Składowa *z* może by pominięta, ponieważ stanowi ona dopełnienie składowych *x* i *y*: $x + y + z = 1$. Przy takim uproszczeniu powstaje pochodna do CIE XYZ przestrzeń oznaczana: CIE xyY, CIE Yxy lub po prostu CIE xy. Wykres chromatyczności powstaje przez przecięcie przestrzeni barw CIE XYZ przez powierzchnię o równaniu $x + y + z = 1$. Otrzymane współrzędne *x*, *y*, *z* nazywane są *współrzędnymi trójchromatycznymi* (nie składowymi!), które zależą jedynie od dominującej długości fali i nasycenia, a nie zależą od luminancji. Wykres ten prezentuje więc wszystkie postrzegane przez człowieka barwy.

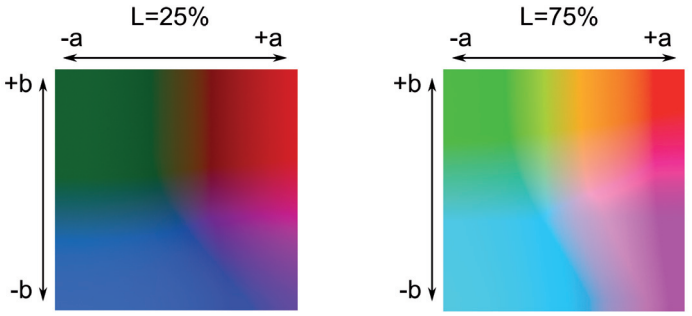
Transformacja odwrotna, z wykresu chromatyczności CIE Yxy do przestrzeni CIE XYZ ma postać:

$$\begin{aligned} X &= \frac{x}{y}Y, \\ Z &= \frac{1-x-y}{y}Y. \end{aligned} \tag{1.7}$$

Wnętrze obszaru ograniczonego przez krzywą barw widmowych wraz z brzegiem reprezentują wszystkie widzialne przez człowieka barwy dla jednej, ustalonej wartości luminancji. Na wykresie nie ma np. brązu, ponieważ jest to barwa pomarańczowa postrzegana przy małej luminancji.

**Równomierne przestrzenie barw**

W modelu CIE XYZ odległości między rozróżnialnymi odcieniami barwy nie są stałe. Dlatego za pomocą matematycznych przekształceń uzyskano przestrzenie, które uznawane są za tzw. *równomierne przestrzenie barwne*. Są to modele CIE Lab oraz CIE LUV. Przykład przestrzeni CIE L*a*b przedstawiono na rysunku 1.23.




**Rys. 1.23.** Przykład przecięcia przestrzeni barw CIE L*a*b w płaszczyźnie chrominancji *ab*, dla różnych wartości luminancji *L*

Równianie (1.9) opisuje transformację współrzędnych z przestrzeni CIE XYZ do percepcyjnie jednorodnej przestrzeni CIE L*a*b:

$$\begin{aligned} L &= 116\sqrt[3]{\frac{Y}{Y_0}} - 16, \\ a &= 500\left(\sqrt[3]{\frac{X}{X_0}} - \sqrt[3]{\frac{Y}{Y_0}}\right), \\ b &= 200\left(\sqrt[3]{\frac{Y}{Y_0}} - \sqrt[3]{\frac{Z}{Z_0}}\right). \end{aligned} \tag{1.8}$$

Współrzędne określone przez punkty: $X_0 = 94{,}81$; $Y_0 = 100{,}0$; $Z_0 = 107{,}3$ są współrzędnymi ciała nominalnie białego. Różnica między barwami wyznaczana jest jako odległość euklidesowa w trójwymiarowej przestrzeni współrzędnych CIE L*a*b:

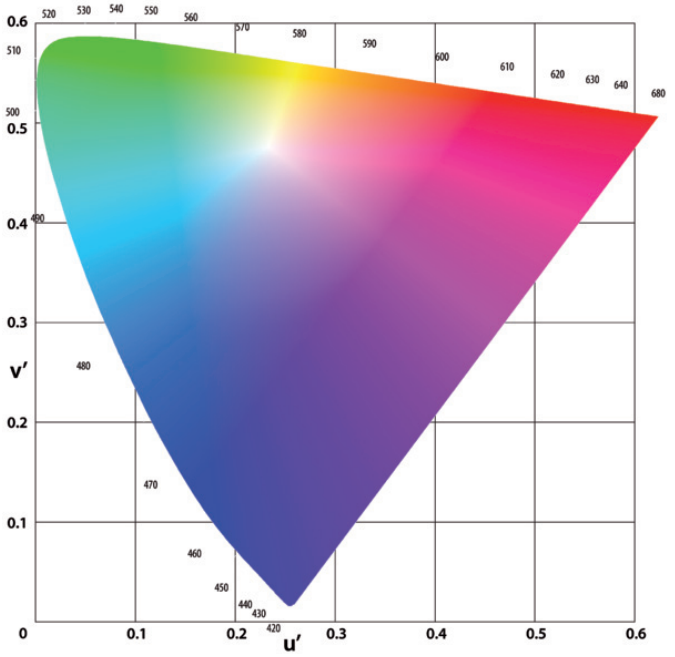
$$\Delta E = \sqrt{(\Delta L)^2 + (\Delta a)^2 + (\Delta b)^2}. \tag{1.9}$$

Tak zdefiniowana różnica barw może posłużyć do określenia wartości, dla której obserwator zauważa różnicę w barwie. Na podstawie badań statystycznych określono $\Delta E$:

$0 < \Delta E < 1$ – obserwator nie zauważa różnicy w barwie,
$1 < \Delta E < 2$ – różnicę zauważa jedynie doświadczony obserwator,
$2 < \Delta E < 3{,}5$ – różnicę zauważa również niedoświadczony obserwator,
$3{,}5 < \Delta E < 5$ – obserwator zauważa wyraźną różnicę barw,
$5 < \Delta E$ – obserwator odnosi wrażenie dwóch różnych barw.

Wartość różnicy $\Delta E$ może zostać wykorzystana do weryfikacji różnych kolorów, zapisanych również w innych przestrzeniach, pod kątem zdolności rozróżnienia. Może to być przydatne np. podczas dobierania zestawu kolorów do tworzonej wizualizacji.

Przykładem przestrzeni percepcyjnie jednorodnej jest również model CIE Luv. Jednak w odróżnieniu od przestrzeni CIE Lab odległość euklidesowa jest różna dla kolorów (płaszczyzna uv) i dla jasności $L$. Na rysunku 1.24 przedstawiono wykres chromatyczności dla przestrzeni CIE Luv.



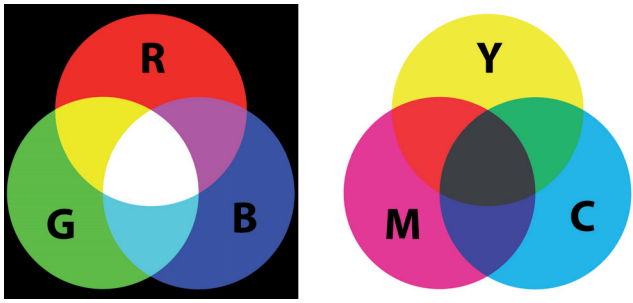

**Rys. 1.24.** Przykład przestrzeni CIE Luv

### 1.3.2. Model RGB

Należy podkreślić, że model RGB oraz model CIE RGB to zupełnie inne podejścia do opisu barwy. Dla przypomnienia: CIE RGB bazował na pomiarach „wrażenia" barwy, zaś przedstawiony poniżej RGB stanowi jedynie „opis działania" urządzeń emitujących światło.

Model RGB wykorzystuje trójchromatyczną teorię widzenia, która zakłada, że dowolną barwę można uzyskać, mieszając trzy odpowiednio dobrane barwy podstawowe. Nazwa RGB powstała z pierwszych liter zastosowanej triady kolorów: czerwonego (ang. *red*), zielonego (ang. *green*) oraz niebieskiego (ang. *blue*). Należy zwrócić szczególną uwagę na fakt, że taki zestaw kolorów został dobrany ze względów technicznych, barwy te zastosowano w luminoforach pierwszych barwnych kineskopów, a obecnie są one stosowane w filtrach ekranów LCD, plazmowych czy też wielkoformatowych LED. Długości fal barw podstawowych będących bazą przestrzeni RGB wynika wyłącznie z właściwości zastosowanych luminoforów czy też filtrów.

Model RGB opisuje addytywne powstawania kolorów, w którym barwy powstają podczas przestrzennego mieszania trzech długości emitowanego światła (rysunek 1.25). Kolor biały uzyskuje się przy maksymalnym wysterowaniu trzech składowych, w przypadku luminoforu odpowiada to jego maksymalnej jasności świecenia.




**Rys. 1.25.** Addytywny model RGB oraz subtraktywne model CMY

Model RGB jest przeważnie modelem dyskretnym, w którym poszczególnym składowym przypisana jest skończona liczba poziomów, wynikająca z przyjętego zapisu (przyjętego sposobu kodowania składowych). Najpopularniejszym formatem jest zapis całkowitoliczbowy, w którym trzy składowe opisują 24 bity, po 8 bitów na składową. Stosuje się również kodowanie zmiennoprzecinkowe, w którym poszczególne składowe przyjmują wartości z przedziału [0,1]. Rozwinięciem standardu RGB jest powszechnie stosowany w grafice komputerowej model RGBA (inne oznaczenia to np. ARGB, BGRA, ARGB24), w którym dodano czwarty kanał (A – ang. *alpha*), określający przeźroczystość. Jest on szczególnie przydatny podczas nakładania wielu obrazów z uwzględnieniem przeźroczystości.

Ograniczenie 8 bitów na jeden kanał wynika z faktu, że model RGB jest ściśle związany ze sprzętowym sposobem generowania obrazów. Pierwsze karty graficzne generowały obraz przez zamianę zawartości pamięci obrazu (ang. *frame buffor*) na sygnały analogowe. Każda składowa (*R*, *G*, *B*) miała własny przetwornik cyfrowo-analogowy o skończonej rozdzielczości. Sygnał analogowy przesyłany do monitora sterował intensywnością pobudzenia

luminoforu. Obecnie sygnał z karty graficznej trafia do monitora drogą cyfrową i dopiero tam zamieniany jest na sygnały analogowe sterujące np. stopniem polaryzacji ciekłego kryształu, która z kolei reguluje wielkością składowych *R*, *G* i *B* piksela w monitorze LCD. Pomimo znacznego postępu technologicznego system 24 bitowego zapisu barwy, dający teoretycznie $3 \times 2^8$ możliwych kolorów, jest nadal stosowany, zapewniając wystarczającą precyzję. Co ciekawe, bardzo często zdarza się, że sprzęt powszechnie stosowany nie jest w stanie odtworzyć poprawnie tak szerokiej palety barw. Dotyczy to przeważnie najtańszych modeli monitorów LCD oraz przeważającej liczby cyfrowych rzutników.

Istnieją jednak zadania, które wymagają większej precyzji w prezentacji koloru, np. obrazowanie medyczne lub profesjonalna obróbka fotografii. Wówczas stosuje się specjalizowane systemy o większej liczbie bitów na kanał, np. 12 bitów w przypadku monochromatycznych, diagnostycznych monitorów medycznych [3].

### 1.3.3. Model CMY

Model ten został stworzony na potrzeby urządzeń drukujących, w których barwa wydruku zależy od ilości pochłoniętego oraz odbitego promieniowania. Jest to model uwzględniający subtraktywne powstawanie barwy, w którym barwa zadrukowanej powierzchni powstaje na skutek pochłonięcia i odbicia widma padającego światła (rys. 1.25). Podobnie jak model RGB, CMY bazuje na trójchromatycznej teorii widzenia, jednak wykorzystuje inny zestaw barw podstawowych: *cyan* (cyjan – odcień koloru niebieskiego), *magenta* (magenta – kolor purpurowy) i *yellow* (kolor żółty). Również w tym modelu stosuje się 8-bitowy zapis poszczególnych składowych.

Uzyskiwana w tym modelu barwa jest bardzo silnie zależna od sprzętu. Jakość wydruku zależy zarówno od zastosowanych atramentów, metody wydruku, powierzchni, na której wykonano wydruk, jak i oświetlenia, w jakim oglądany jest wydruk.

Równanie (1.10) opisuje przejście między modelem RGB oraz CMY:

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \tag{1.10}$$

Teoretycznie możliwe jest uzyskanie barwy czarnej poprzez zmieszanie triady podstawowej (*C*, *M*, *Y*). Jednak uzyskiwana w ten sposób czerń nie jest zadowalająca – zazwyczaj postrzegana jest jako bardzo ciemny odcień barw podstawowych. Dodanie czwartego koloru w postaci „czysto czarnego" atramentu rozwiązało problem „brudnej" czerni na wydrukach. Barwa czarna powstaje (teoretycznie) na skutek zmieszania w równych proporcjach barw podstawowych, tak więc równanie (1.11) opisuje przejście z modelu CMY do CMYK:

$$\begin{gathered} K = min(C', M', Y'), \\ \begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} C' \\ M' \\ Y' \end{bmatrix} - \begin{bmatrix} K \\ K \\ K \end{bmatrix}. \end{gathered} \tag{1.11}$$

[3] Monitor EMD213 `http://www.medicaldisplaypro.com/` lub Eizo GS520

Należy zwrócić uwagę, że powyższe składowe kolorów podstawowych (czyli $R,G,B$ oraz $C,M,Y,K$) nie są ściśle określone – luminofor ekranów czy też atramenty różnych producentów mogą być postrzegane w różnych sposób. Problem ten zostanie szerzej omówiony w kolejnych rozdziałach, a w szczególności w rozdziale 1.3.7.

## 1.3.4. Modele HSV i HSL

Modele te powstały przez przekształcenie kartezjańskich współrzędnych modelu RGB do współrzędnych cylindrycznych. Są to modele dostosowane do sposobu postrzegania koloru przez człowieka i szeroko używane, głównie dzięki bardziej intuicyjnemu sterowaniu w porównaniu z modelem RGB. Przestrzeń w tych modelach ma trzy współrzędne: barwę (ang. *Hue*), nasycenie koloru (ang. *Saturation*) oraz jasność (ang. *Value* lub *Lightness*). Jak widać, każda współrzędna odpowiada za jedną, konkretną wielkość, której zmiana w sposób jednoznaczny wpływa na opisywany kolor. Wpływ zmian współrzędnych w modelu RGB nie jest tak oczywisty. Na rysunku 1.26 przedstawiono sposób przejścia z modelu RGB do modeli HSV oraz HSL. Po „odwróceniu" przestrzeni RGB w taki sposób, by oś pionowa pokrywała się z barwami achromatycznymi (od czerni do bieli), przesuwa się wszystkie barwy podstawowe (wierzchołki RGBCMY) na jedną płaszczyznę. Płaszczyzna ta w przypadku modelu HSV umieszczana jest na poziomie bieli, a w modelu HSL w połowie odległości między bielą i czernią. Uzyskany odwrócony ostrosłup modelu HSV oraz podwójny ostrosłup modelu HSL umieszczane są w cylindrycznym układzie współrzędnych.

Współrzędne opisujące saturację ($S$), wartość ($V$) oraz jasność ($L$) przyjmują wartości od 0 do 100%, zaś współrzędna odcienia barwy ($H$) od 0 do 360°. Bardzo często, zamiast wartości procentowych stosuje się zapis w postaci liczb rzeczywistych z przedziału od 0 do 1.

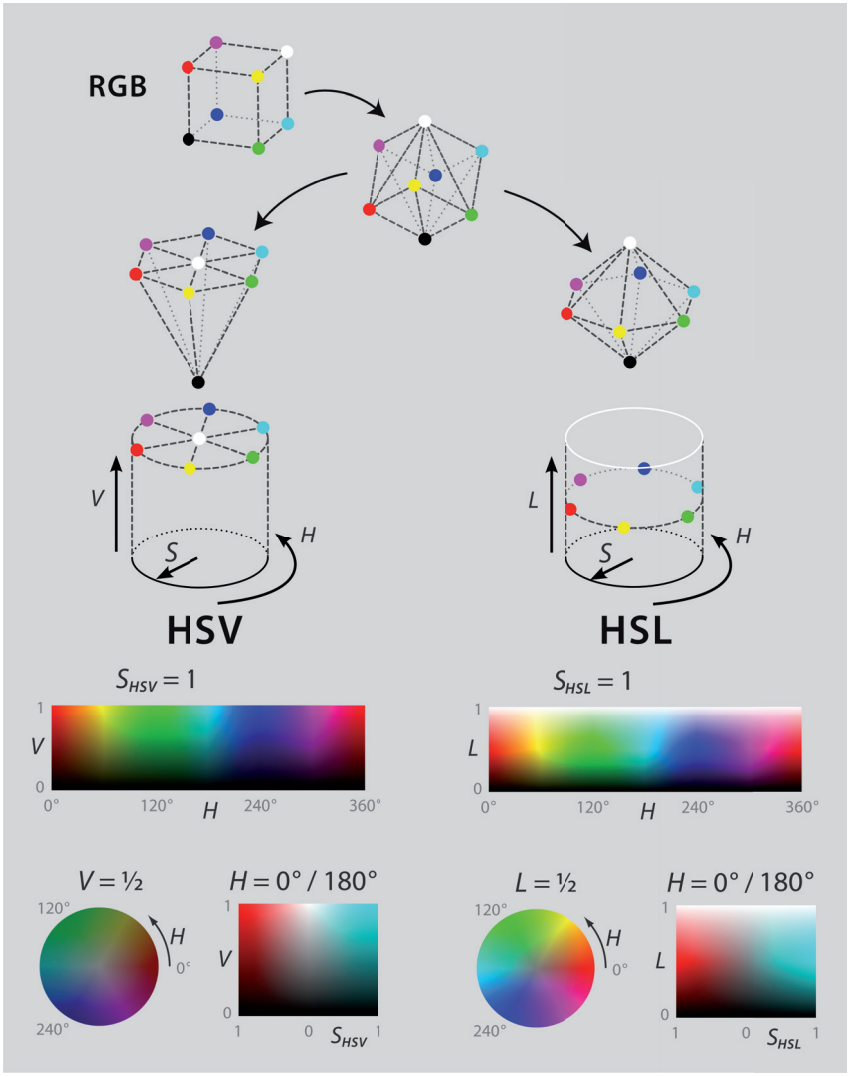
Przestrzeń HSV naśladuje proces tworzenia barwy podczas malowania farbami. Najpierw wybierany jest odcień barwy (współrzędna $H$), następnie poprzez dodanie „białej farby" ($S$) zmniejszone zostaje nasycenie zaś poprzez dodanie „czarnej farby" ($V$) ustalana jest jasność.

Przestrzenie HSV oraz HSL nie mają cech przestrzeni liniowej, nie obowiązuje w nich prawo dodawania wektorów, tak więc nie istnieje macierzowe przejście między tymi przestrzeniami a przestrzeniami mającymi charakter liniowy (jak np. RGB).

**Konwersje RGB-HSV i RGB-HSL**

Konwersję z przestrzeni RGB do HSV opisują równania (1.12). Zakłada się, że składowe $R$, $G$ i $B$ zawierają się w przedziale od 0,0 do 1,0.

$$\begin{gathered} MAX = max(R,G,B), \\ MIN = min(R,G,B), \\ H = \begin{cases} (0 + \dfrac{G-B}{MAX-MIN}) \cdot 60, & \text{if } R = MAX \\ (2 + \dfrac{B-R}{MAX-MIN}) \cdot 60, & \text{if } G = MAX \\ (4 + \dfrac{R-G}{MAX-MIN}) \cdot 60, & \text{if } B = MAX \end{cases}, \\ S = \frac{MAX - MIN}{MAX}, \\ V = MAX. \end{gathered} \tag{1.12}$$




**Rys. 1.26.** Powstawanie przestrzeni HSV oraz HSL z modelu RGB (na podstawie grafik Jacob Rus, Wikipedia)

W wyniku obliczeń $S$ i $V \in [0,0; 1,0]$ zaś $H \in [0,0; 360,0]$. Jeżeli wartość maksymalna i minimalna dla składowych RGB są sobie równe ($MAX = MIN$), wówczas $H$ jest niezdefiniowane. Ma to miejsce w przypadku, gdy nasycenie koloru jest zerowe, czyli $S = 0$. Wówczas rozpatrywany kolor w przestrzeni RGB jest achromatyczny (poziom szarości), czyli nie można mu przypisać konkretnej wartości odcienia $H$. Podobnie w przypadku czerni, gdy $MAX = 0$ oraz $V = 0$, składowa saturacji $S$ nie jest zdefiniowana.

W przypadku przestrzeni HSL, współrzędne *H* obliczane są z RGB w taki sam sposób, jak dla przestrzeni HSV (1.12). Różnice występują dla składowej nasycenia (*S*) oraz jasności (*L*). Równanie (1.13) opisują współrzędne *S* i *L* dla modelu HSL:

$$\begin{array}{c} S = MAX - MIN, \\ L = \frac{MAX+MIN}{2}. \end{array} \tag{1.13}$$

Konwersja barw z przestrzeni HSV do RGB może być wykonana za pomocą następujących zależności (1.14):

$$\begin{array}{c} H_i = \left\lfloor \frac{H}{60} \right\rfloor, \\ f = \frac{H}{60} - H_i, \\ p = V(1-S), \\ q = V(1-fS), \\ t = V(1-(1-f)S), \\ \text{if } H_i = 0 \rightarrow R = v, \quad G = t, \quad B = p \\ \text{if } H_i = 1 \rightarrow R = q, \quad G = v, \quad B = p \\ \text{if } H_i = 2 \rightarrow R = p, \quad G = v, \quad B = t \\ \text{if } H_i = 3 \rightarrow R = p, \quad G = q, \quad B = v \\ \text{if } H_i = 4 \rightarrow R = t, \quad G = p, \quad B = v \\ \text{if } H_i = 5 \rightarrow R = v, \quad G = p, \quad B = q \end{array} \tag{1.14}$$

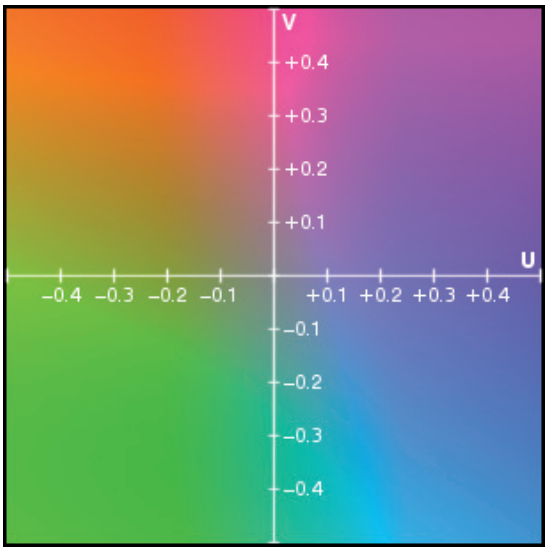
W wyniku obliczeń uzyskuje się wartości RGB znormalizowane do przedziału [0,0; 1,0].

Obydwa modele są często wykorzystywane w programach graficznych, ponieważ udostępniają intuicyjne sterowanie barwą. Inną istotną cechą jest proste tworzenie przejść między kolorami poprzez liniową interpolację współczynników modelu. Dla przykładu, interpolując liniowo jedynie współrzędną *H* od wartości 0 przez 120 do 240, uzyskuje się tęczę. Tak proste sterowanie kolorem jest szczególnie istotne przy tworzeniu tablic kolorów używanych do wizualizacji, o czym będzie mowa w dalszej części tej pracy.

### 1.3.5. Model YUV

Standard ten powstał na potrzeby przejścia od czarno-białej telewizji analogowej do telewizji kolorowej nadającej w standardzie PAL. Ponieważ pierwsze systemy telewizyjne rejestrowały, przesyłały i wyświetlały jedynie jasność obrazu, podczas wprowadzania sygnału „kolorowego" zadbano o wsteczną kompatybilność. Sygnał luminancji *Y* został uzupełniony dwoma kanałami o barwie: kanał U stanowił przeskalowaną składową niebieską B z modelu RGB, zaś kanał V przeskalowaną składową czerwieni R. Dzięki temu, starsze odbiorniki mogły nadal pracować z nowym sygnałem. Dodatkowo, model YUV uwzględnia przeciwstawny sposób postrzegania barw przez ludzki mózg w postaci dwóch kanałów chrominancji *r/g* i *y/b* oraz kanału luminancji *L*. Zagadnienia te omówiono w rozdziale 1.2.3.

Na rysunku 1.27 przedstawiono przestrzeń kolorów wyznaczoną przez współrzędne U oraz V.




**Rys. 1.27.** Przestrzeń modelu YUV dla maksymalnej wartości współrzędnej $Y$

Model YUV jest liniową transformacją modelu RGB, którą opisuje następująca zależność (1.15).

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ -0{,}147 & -0{,}289 & 0{,}437 \\ 0{,}615 & -0{,}515 & -0{,}100 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}. \tag{1.15}$$

Równanie (1.15) jest szczególnie cenne, ponieważ umożliwia obliczenie poprawnej percepcyjnie „jasności" dla obrazów kolorowych. Ma to szczególne znaczenie przy tworzeniu wizualizacji, które będą prezentowane zarówno w postaci kolorowej grafiki, jak i w postaci monochromatycznych wydruków.

Standard YUV jest obecnie używany w postaci wielu modyfikacji. Przykładem jest Y'UV uwzględniający kompensację kanału luminancji o współczynnik gamma. Z kolei odmiana YPbPr stanowi podstawę profesjonalnego standardu do przesyłu analogowego sygnału wideo o wysokiej rozdzielczości tzw. *Component Video*. YCbCr jest wykorzystywany w telewizji cyfrowej, szczególnie do kompresji sygnału wideo. W tym wypadku korzysta się z faktu, że ludzkie oko jest bardziej czułe na utratę informacji o jasności niż o barwie. Dlatego różnicowe kanały chrominancji mogą być kompresowane stratnie.

## 1.3.6. Model achromatyczny

Światło achromatyczne daje wrażenie światła białego. Jego maksymalna jasność w danych warunkach odpowiada barwie białej. Natomiast brak strumienia światła to barwa czarna. Wartości pośrednie stanowią poziomy (nie odcienie!) szarości. Przyjmując, że czerni przypi-

szemy wartość 0, zaś bieli 1, poziomy pośrednie mogą być wyznaczone w sposób praktycznie dowolny. Może to być podział równomierny, jednak nie będzie on postrzegany jako liniowa zmiana jasności. Ludzkie oko bardziej jest czułe na stosunek wartości strumienia niż na jego wartości bezwzględne. Z tego powodu stosuje się skalę logarytmiczną zamiast liniowej.

Z przeprowadzonych badań wynika, że ludzkie oko nie jest w stanie rozróżnić dwóch sąsiadujących wartości natężenia światła, których stosunek jest niniejszy od $I_{j+1}/I_j \leq 1{,}01$ [Zab94].

Dodatkowo same urządzenia, takie jak: monitory, skanery czy też aparaty fotograficzne wprowadzają swoje własne nieliniowości do rejestrowanych lub odtwarzanych obrazów. W celu kompensacji nieliniowej percepcji jasności wykorzystuje się tzw. korekcję gamma. Polega ona na takim przeliczeniu wartości jasności, by postrzegana była jako liniowa. Równanie (1.16) opisuje zależność między luminancją wejściową a wyjściową, z uwzględnieniem współczynnika korekcji γ:

$$L_{wyj} = L_{wej}^{\gamma}. \tag{1.16}$$

Współczynnik korekcji γ dla monitorów powinien wynosić około 2,2. Wartość ta została przyjęta za „standardową" dla komputerów klasy PC. Dostępne wartości korekcji w monitorach wahają się od wartości 1,4 do 2,8.

Szczególnego znaczenia nabiera właściwa kalibracja monitorów medycznych, czyli takich, które są używane do prezentacji monochromatyczne danych medycznych. Przykładem mogą być mammogramy lub inne dane radiologiczne wymagające analizy np. bardzo niewielkich różnic w intensywności. Wizualizacja takich danych na monitorze nieskalibrowanym lub skalibrowanym niepoprawnie może prowadzić do poważnych błędów zagrażających zdrowiu a nawet życiu pacjenta – monitor może nie wyświetlać danych istotnych medycznie. Zasady kalibracji i konfiguracji monitorów medycznych zostały opisane w dokumentacji standardu DICOM w postaci specyfikacji GSDF (ang. *Grayscale Standard Display Function*) oraz GSPS[4] (ang. *Grayscale Softcopy Presentation State*).

## 1.3.7. Zarządzanie informacją o kolorze

Modele barw RGB oraz CMYK powstały na potrzeby opisu zjawisk powstawania barwy zachodzących w konkretnych typach urządzeń. Jednak różni producenci monitorów lub drukarek stosują inne odcienie barw podstawowych, co prowadzi do tego, że ten sam cyfrowo zapisany obraz może wyglądać zupełnie inaczej na różnych wyświetlaczach, rzutnikach czy też wydrukach. Dlatego niezbędna okazała się standaryzacja, zapewniająca powtarzalność otrzymywanych wrażeń barwnych.

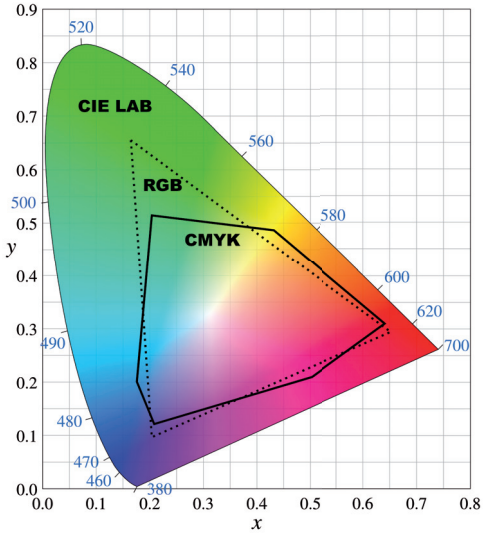
Przemysł poligraficzny rozwiązał problemem standaryzacji opisu barwy poprzez wprowadzenie profili barwnych. Profil barwy to zbiór danych, który w postaci pliku opisuje zdolność rejestrowania lub odtwarzania barwy. Organizacja ICC[5] (ang. *International Color Consortium*) określiła zasady oraz zawartość plików z profilami barwnymi. Dlatego do określ-

[4] Dokument PS 3.14 standardu DICOM.

[5] International Color Consortium, `http://www.icc-cms.com/`

lenia profilu barwnego stosuje się skrót ICC. Dane zawarte w tych plikach umożliwiają konwersję przestrzeni kolorów zależnych od urządzeń (np. RGB monitora, rzutnika i CMYK drukarki czy laboratorium fotograficznego) do jednej z przestrzeni niezależnych od urządzenia (percepcyjnych CIE) zazwyczaj CIE L*a*b lub CIE XYZ.

Każde urządzenie ma pewien ograniczony zakres barw, jakie może zarejestrować lub wygenerować. Dla przykładu monitor nie może wyświetlić koloru „bardziej czerwonego", niż ten jaki powstaje przy maksymalnym wysterowaniu kanału czerwonego. Taki podzbiór barw będący wycinkiem przestrzeni wszystkich kolorów rozróżnialnych przez człowieka nazywany jest *gamutem*. Przykład teoretycznych gamutów ekranu monitora oraz drukarki przedstawiono na rysunku 1.28.



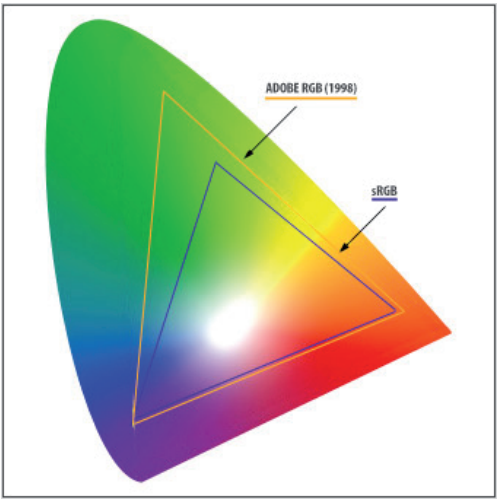

**Rys. 1.28.** Gamut ekranu (model RGB) oraz drukarki (model CMYK) wrysowany w wykres chrominancji przestrzeni CIEYxy, czyli przestrzeń wszystkich postrzeganych przez człowieka barw.

Ponieważ przestrzenie barw różnych urządzeń, takich jak: cyfrowe aparaty fotograficzne, skanery, monitory czy drukarki, nie pokrywają się ze sobą, aby zapewnić powtarzalność niezbędne są konwersje między przestrzeniami barw tych urządzeń. Konwersje te można wykonać na kilka sposobów. Można uwzględniać wrażenia percepcyjne (zachować wrażenie barwy), gdy szczególnie dużo barw znajduje się poza gamutem. Inną metodą jest zachowanie maksymalnego nasycenie kolorów lub użycie metod kolorymetrycznych: relatywnej i absolutnej. W metodzie relatywnej porównuje się najjaśniejsze kolory gamutu źródłowego oraz docelowego, a następnie przesuwa wszystkie kolory tak, by mieściły się w nowej przestrzeni barw. Z kolei metoda absolutna przenosi kolory z gamutu źródłowego do docelowego bez przesunięć, odcinając barwy, które nie występują w gamucie docelowym. Jest to metoda przydatna np. przy symulowaniu wyników działania urządzeń CMYK na ekranie RGB.

W celu uniknięcia nadmiernej liczby konwersji, a także aby uprościć zarządzanie kolorami w nieprofesjonalnych systemach komputerowych, opracowano standard sRGB. Pokrywa on gamut wielu urządzeń: zarówno ekranów, jak i drukarek, skanerów, aparatów fotograficznych itp. Osiągnięto to poprzez zawężenie zakresów możliwych barw np. wyświetlanych na monitorze LCD. Niestety, standard ten równa do najuboższych urządzeń, bo tego wymaga wspólna przestrzeń dla dowolnych urządzeń. sRGB pokrywa jedynie około 35% przestrzeni CIE XYZ. Standard sRGB zakłada 256 poziomów poszczególnych składowych, co daje teoretyczną liczbę 16 milionów kolorów. Liczba ta nie ma jednak większego znaczenia wobec faktu, że dość drastycznie ograniczone zostały maksymalne intensywności kolorów. Można to zauważyć porównując rysunki 1.28 i 1.29. W przestrzeni sRGB wprowadzono szczególnie duże ograniczenia w zakresie barw zielonych.

Model sRGB jest przestrzenią barw niezależną od sprzętu, tak jak przestrzeń CIE XYZ. Pomostem między przestrzeniami zależnymi od sprzętu (np. RGB, CMYK) a sRGB są profile ICC konkretnych urządzeń. Model sRGB został uznany za standardowy dla dokumentów HTML sieci WWW oraz telewizji wysokiej rozdzielczości HDTV.

Innym przykładem przestrzeni niezależnej od sprzętu, a stworzonej do ułatwienia zarządzania kolorami, jest AdobeRGB. Przestrzeń ta zawiera większość kolorów możliwych do uzyskania na wydrukach, ale przy założeniu, że można je wyświetlić na monitorze. Przestrzeń ta pokrywa około 50% widzianych przez człowieka barw.




**Rys. 1.29.** Porównanie przestrzeni barw sRGB oraz AdobeRGB.

Poprawna konfiguracja sprzętu (np. temperatury barwy ekranu monitora) oraz oprogramowania umożliwia uzyskanie na wydrukach barw bardzo zbliżonych do tych widzianych na ekranie monitora. Warto sobie również zdawać sprawę z tego, że powszechnie stosowane podczas prezentacji cyfrowe rzutniki mają z reguły znacznie ograniczony gamut. Istnieją również rzutniki, szczególnie wykonane w technologi DLP, które mają bardzo szeroki zakres

reprezentowanych kolorów. Również zastosowanie podświetlenia LED w ekranach LCD poszerza gamut wyświetlacza.

Profil kolorów ICC można uzyskać od producenta oraz przez kalibrację. Przedstawione zagadnienia dotyczące zasad zarządzania i kalibracji kolorów zostały przedstawione w sposób skrótowy. Osoby zainteresowane tą tematyką zachęcam do lektury licznych publikacji oraz książek z tego zakresu, np. książki „Profesjonalne zarządzanie barwą", której autorami są: Bruce Fraser, Chris Murphy, Fred Bunting.

## 1.4. Zdolność analizy obrazu

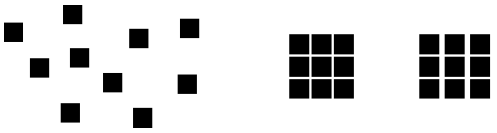
Podstawy teorii percepcji wzrokowej opracowane zostały przez niemieckich psychologów (Max Wertheimer, Kurt Koffka oraz Wolfgang Köhler) w latach 20. XX wieku. Opracowana przez nich teoria nazwana „Gestalt", z niemieckiego „kształt" lub „forma", dotyczy *relacji pomiędzy całością a częściami*, z których się składa. Analizuje jak całość i jej części są postrzegane przez umysł. Obecnie termin „Gestalt" odnosi się głównie do szkoły terapii w psychologii (psychoterapii) oraz kierunku psychologii, zaś do opracowanej teorii percepcji odnosi się termin *psychologia postaci.*

Podstawowe przesłanie tej teorii zawiera się w zdaniu: *Całość to coś więcej niż suma jej elementów.* Teoria „Gestalt" starała się opisać, w jaki sposób ludzie porządkują i organizują postrzegane elementy. W przypadku percepcji wzrokowej teoria ta zakłada, że obraz jest czymś więcej niż tylko sumą elementów graficznych (punktów, linii, kształtów, kolorów itp.). Znajomość podstaw tej teorii może ułatwić przewidywanie, w jaki sposób odbiorca zareaguje na elementy składowe, a tym samym na cały przekaz wizualny. Uwzględnianie poniższych praw teorii percepcji wzrokowej ułatwi tworzenie zrozumiałych przekazów.

### 1.4.1. Prawa teorii percepcji wzrokowej

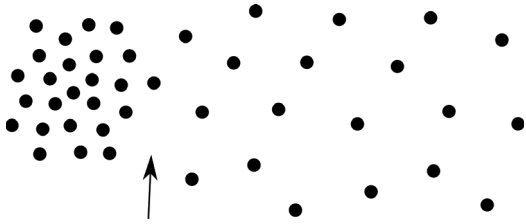
Badania przeprowadzone przez psychologów teorii „Gestalt" są współcześnie nadal cenione, ponieważ opisują w prosty sposób wiele podstawowych zjawisk towarzyszących percepcji obrazu. Opracowane szczegółowe zasady, tzw. *prawa,* opisują w jaki sposób widzimy wzorce wizualne.

**Bliskość** jest bardzo silnie postrzegana oraz interpretowana, szczególnie gdy odległości między elementami są relatywnie małe. Na rysunku 1.30 przedstawiono przykłady trzech grup obiektów, które w zależności od odległości między nimi postrzegane są jako grupa osobnych niezależnych obiektów (po lewej), bądź jako elementy postrzegane razem.



**Rys. 1.30.** Wpływ bliskości obiektów na postrzeganie elementów jako obiektów niezależnych bądź jako grupa

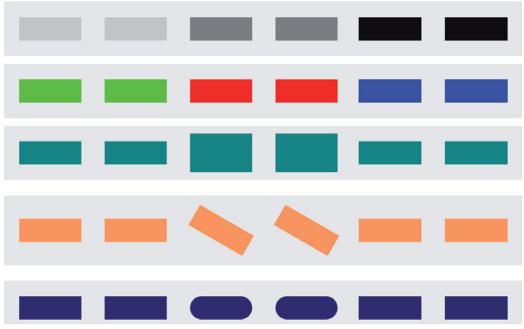
Pomimo podobnej liczby obiektów i identycznych kształtów to *bliskość* poszczególnych elementów zmienia zupełnie sposób postrzegania całości. Nawet niewielkie zmiany odległości między elementami są wychwytywane i interpretowane jako zmiana przynależności (rysunek po prawej).
Jednak odległość nie jest jedynym czynnikiem postrzegania i przewidywania przynależności elementów do grup. Poza odległością brana jest również pod uwagę przestrzenna koncentracja elementów – regiony o podobnej gęstości elementów traktowane są jako grupa. Przykładem takiego postrzegania koncentracji jest zaklasyfikowanie obiektu wskazywanego przez strzałką. Punkt ten postrzegany jest jako część grupy o większej gęstości (rys. 1.31).



**Rys. 1.31.** Postrzeganie koncentracji: punkt wskazywany strzałką postrzegany jest jako część grupy o większej koncentracji

Zastosowanie własności bliskości w grafice jest bardzo proste: najprostszym i najefektywniejszym sposobem podkreślenia związków między danymi jest umieszczenie symboli je reprezentujących blisko siebie.

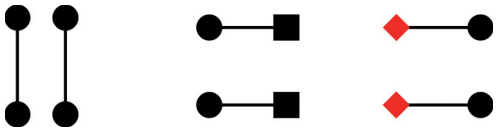
**Podobieństwo** stanowi bardzo łatwo interpretowaną własność. Występuje wówczas, gdy możemy wyróżnić w obiektach cechy wspólne. Obiekty podobne postrzegane są jako grupy lub wzory. Rysunek 1.32 przedstawia grupy obiektów będące przykładem obiektów spełniających zasadę podobieństwa.



**Rys. 1.32.** Zasada podobieństwa powoduje, że postrzegamy w każdym wierszu trzy grupy po dwa elementy

Podobieństwo może dotyczyć rożnych atrybutów graficznych. Na rysunku 1.32 w kolejnych wierszach wykorzystano podobieństwo: jasności, koloru, rozmiaru, orientacji oraz kształtu. W każdej linii, dzięki podobieństwu, rozróżniane są trzy grupy po dwa elementy. Co warto szczególnie podkreślić, różne rodzaje podobieństwa są przeważnie dobrze rozróżniane. Dzięki temu, każdy rodzaj podobieństwa może stanowić oddzielne kryterium klasyfikacji przynależności konkretnego obiektu do różnych grup.
W grupie elementów podobnych szczególnie widoczne są obiekty niepodobne do innych, które można nazwać anomaliami. Elementy takie szczególnie silnie, podświadomie przyciągają uwagę.

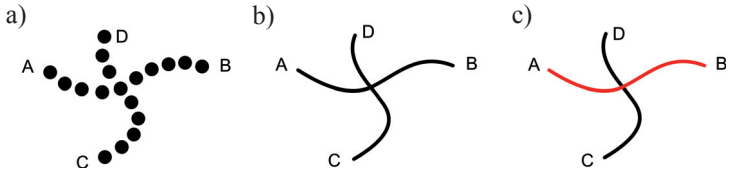
**Powiązanie** nie zostało odnotowane przez twórców teorii „Gestalt", jednak stanowi bardzo silnie oddziałujący przekaz. Na rysunku 1.33 zaprezentowano przykłady elementów powiązanych za pomocą odcinków linii.



**Rys. 1.33.** Powiązanie elementów jest najsilniejszym sposobem grupowania, silniejszym niż odległość, podobieństwo kształtu czy koloru

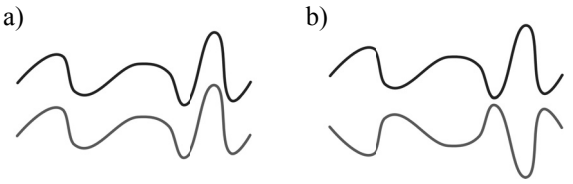
Niezależnie od zastosowanych grupowań, za pomocą bliskości obiektów, podobieństwa kształtu czy koloru, obiekty połączone liniami zawsze postrzegane są jako grupy. Tak silne oddziaływanie wykorzystywane jest powszechnie w diagramach do łączenia różnych elementów.

**Ciągłość** występuje, gdy oko podświadomie zmuszane jest do „poruszania" się w konkretnym kierunku lub po konkretnych obiektach. Ludzki umysł ma tendencję do grupowania rzeczy, które wskazują płynny kierunek ruchu, bez gwałtownych zmian. Przykładem postrzegania ciągłości może być rysunek 1.34 a), na którym zaprezentowano szereg elementów tworzących „ścieżki" między literami A, B, C oraz D. Ścieżki te wyodrębniane są ze względu na zasadę podobieństwa oraz bliskości i odbierane w taki sam sposób, jak krzywe przedstawione na rysunku 1.34 b). Dodatkowo dzięki zasadzie ciągłości, odcinek AB, zaznaczony kolorem czerwonym na rysunku 1.34 c), postrzegany jest jako „bardziej naturalna" droga, niż np. ścieżka AC lub AD.



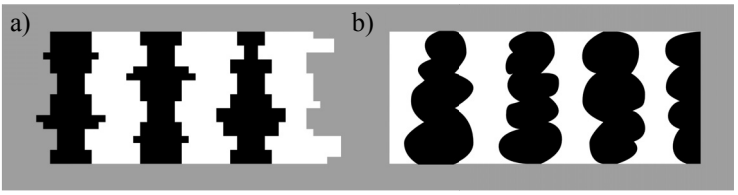

**Rys. 1.34.** Przykład postrzegania ciągłości

**Symetria** wywiera silne wrażenie, którego skutkiem jest grupowanie elementów symetrycznych. Rysunek 1.35 przedstawia dwie krzywe, raz postrzegane jako dwa niezależne obiekty (1.35a) oraz jako jeden (1.35b), gdy występuje symetria.




**Rys. 1.35.** Przykład zastosowania symetrii do porównania kształtów krzywych

Na rysunku 1.36a) zaprezentowano przykład trzech czarnych oraz białych figur. Czarne kształty postrzegane są jako oddzielne obiekty na białym tle. Dzieje się tak, ponieważ czarne figury są symetryczne. Do *symetrii* zalicza się również zasadę wypukłości, która głosi że obiekty wypukłe, a nie wklęsłe, mają tendencję do bycia postrzeganymi jako całość lub zgrupowanie. Przykładem tej zasady jest rysunek 1.36b), na którym widzimy czarne kształty na białym tle, a nie odwrotnie.




**Rys. 1.36.** Przykład postrzegania symetrii oraz wypukłości

**Zamknięcie** pojawia się, gdy obiekt jest niekompletny lub nie jest całkowicie zamknięty. Jeżeli kształt jest wystarczająco określony, wówczas całość postrzegana jest jako kształt zamknięty. Przykładem może być rysunek 1.37.

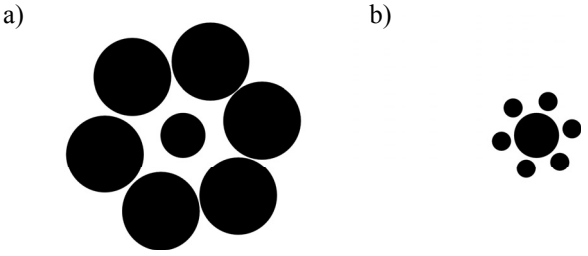
**Rys. 1.37.** Przykłady percepcyjnego zamykania kształtów

Mimo że rysunek pandy składa się jedynie z kilku plam, a górna część głowy oraz grzbietu nie są bezpośrednio zaznaczone, panda postrzegana jest jako kształt zamknięty. Również na drugim rysunku „widzimy" kolczastą kulę, chociaż nie ma tam jej faktycznej reprezentacji. Kluczowe znaczenie dla zjawiska *zamykania* ma pamięć obrazów oraz doświadczeń – kształt pandy jest bardzo dobrze znany, dlatego w kilku „plamach" rozpoznajemy tułów oraz głowę sympatycznej pandy.

**Figura i tło** są rozróżniane jako odrębne części obrazu poprzez różnice nie tylko jasności i koloru, ale również wielu innych cech, takich jak: kształt, tekstura. Figury postrzegane są zazwyczaj „z przodu", zaś tło „z tyłu". Dodatkowo granica rozdzielająca obiekt (figurę) i tło przypisywana jest do obiektu. Tło postrzegane jest zazwyczaj jako część o mniejszej wadze percepcyjnej. Rysunek 1.38 przedstawia dwie takie same figury umieszczone na jasnym oraz ciemnym tle. W zależności od zastosowanych kolorów, można zauważyć twarz kobiety lub saksofonistę.

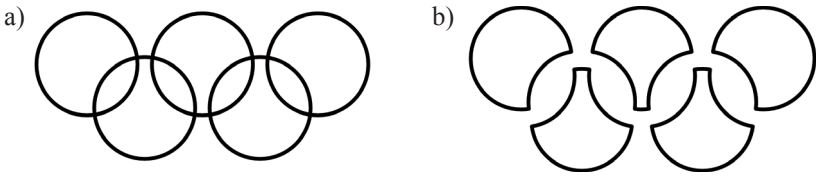
**Rys. 1.38.** Tło oraz figura mogą być różnie interpretowane, zależnie np. od koloru

**Otoczenie** ma znaczący wpływ na postrzeganie rozmiarów i stanowi przykład wzmacniającego działania kontrastu. Efekt ten można zaobserwować na rysunku 1.39. W obu przypadkach środkowe koło ma identyczny rozmiar, ale jego postrzeganie zależy od wielkości otaczających je obiektów. Sąsiedztwo dużych kół powoduje „zmniejszanie się" środkowego obiektu, zaś sąsiedzi o małych rozmiarach powodują pozorne jego zwiększanie.




**Rys. 1.39.** Wpływ otoczenia na postrzeganie wielkości środkowego koła

**Zwięzłość** – obraz postrzegany jest w sposób najprostszy z możliwych. Symbol olimpiady widziany jest jako pięć okręgów (rysunek 1.40a), a nie jako zbiór łuków (rysunek 1.40b)




**Rys. 1.40.** Przykład zwięzłości postrzegania – symbol a) postrzegany jest jako zbiór okręgów, a nie jako zbiór kształtów zaprezentowanych na rysunku b)

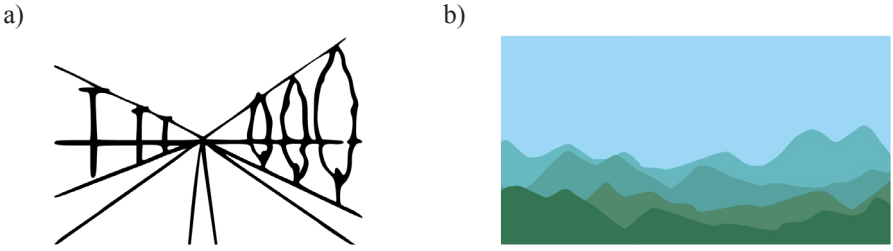
Przytoczone powyżej *prawa psychologii postaci* mogą wydawać się oczywiste, jednak uświadomienie sobie tych mechanizmów postrzegania obrazu może uchronić przed tworzeniem „fałszywych wizualizacji", czyli takich przekazów graficznych, które są zupełnie inaczej, „opacznie" postrzegane przez odbiorcę. Łamania zasad postrzegania obrazów można się doszukać w specjalnie przygotowywanych obrazach ukazujących złudzenia optyczne.

## 1.4.2. Postrzeganie przestrzeni

Człowiek posiada dwoje oczu i dzięki temu może rozpoznawać odległości oraz inne przestrzenne zależności między postrzeganymi obiektami. Jednak samo wrażenie przestrzenności mogą wywołać nawet obrazy dwuwymiarowe.

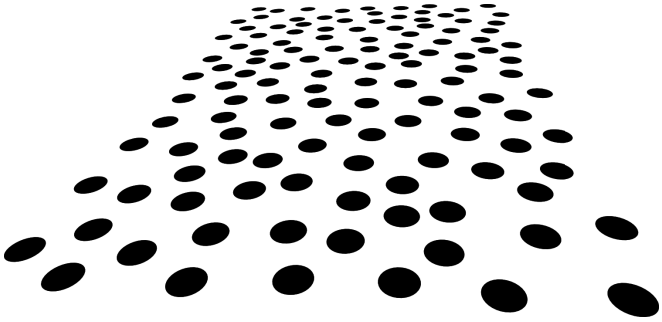
Poniżej znajduje się lista cech obrazu, na podstawie których postrzegana jest przestrzeń. Zostały one podzielone na trzy kategorie:

1. Przestrzeń na obrazach statycznych:
   - Zbieżna perspektywa – linie zbiegające się w jednym punkcie tworzą złudzenie przestrzenności. Efekt ten jest wynikiem doświadczeń i obserwacji obiektów w rzeczywistym świecie. Równoległe tory kolejowe postrzegane są jako proste zbiegające się w jednym punkcie na horyzoncie. Przykładem perspektywy o jednym punkcie zbieżności jest rysunek 1.41a). Sposób postrzegana rozmiarów i przestrzeni zależy od występującej w obrazie liczby punktów zbieżności.



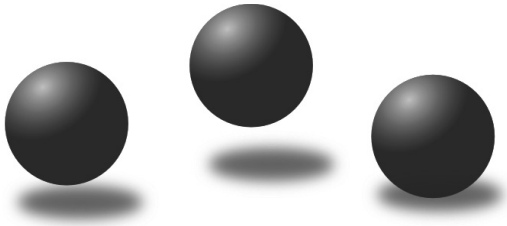

**Rys. 1.41.** Przykłady postrzegania przestrzeni: a) zbieżna perspektywa oraz b) perspektywa powietrzna

- Perspektywa powietrzna – wraz ze wzrostem odległości, szczególnie przy obserwacji rozległych obszarów (np. gór), kolory blakną oraz nabierają odcienia niebieskiego. Dzieje się tak w rezultacie rozpraszanie światła w powietrzu. Rysunek 1.41b przedstawia kilka kształtów, w których można doszukać się szeregu łańcuchów górskich.
- Gradient tekstury – zmiana rozmiarów, częstotliwości czy proporcji regularnych wzorów w rzeczywistym świecie, świadczy o zmianie odległości lub zmianie kształtu, może więc być nośnikiem informacji o głębi lub wzajemnych odległościach między obiektami (rysunek 1.42).

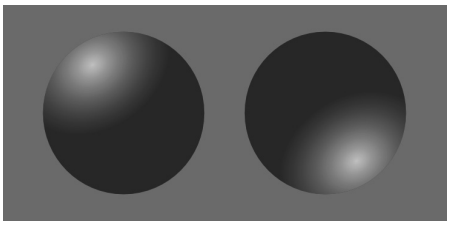


**Rys. 1.42.** Przykład gradientu tekstury, wzór interpretowany jest jako płaska powierzchnia np. stołu

- Gradient rozmiarów – obiekty oddalające się zmniejszają swoje rozmiary Poprawna interpretacja rozmiarów często wymaga tzw. „punktu odniesienia", czyli elementu o znanych rozmiarach, do którego można porównać inne obiekty.
- Otoczenie – sposób postrzegania poszczególnych fragmentów zawsze jest związany z całością, szczególnie, gdy występują wzajemne relacje np. wzajemne przesłanianie lub rzucanie cieni.
- Głębia ostrości – nieostrość części obrazu jest naturalną cechą układu optycznego, w tym oka. Gdy patrzymy na obiekty bardzo oddalony, wówczas obiekty najbliższe są rozmyte.
- Cienie – cienie rzucane przez obiekty są bardzo wyraźnym sygnałem o przestrzennym położeniu, na rysunku 1.43 przedstawiono przykład w jaki sposób obecność cienia wpływa na sposób interpretacji odległości między kulami a teoretycznym podłożem.



**Rys. 1.43.** Wpływ cieni na postrzeganie odległości między obiektami



**Rys. 1.44.** Wpływ oświetlenia na postrzeganie kształtu

- Kształt wynikający z oświetlenia – przestrzenny kształt obiektu można bardzo dokładnie rozpoznać po sposobie w jaki odbija i rozprasza światło. Na rysunku 1.44 przedstawiono dwa ciemne koła, z nałożonym jasnym gradientem, który interpretowany jest jako rozbłysk światła. Mózg interpretuje lewe koło jako wypukłe, zaś prawe jako wklęsłe, ponieważ „przyzwyczajony jest" do oświetlenia padającego z góry.
- Akomodacja oka (jest niezależna od obrazu) – zjawisko to występuje, gdy oko „zmienia nastawy", zarówno ogniskowej soczewki, czyli zmienia odległość ostrego widzenia jak i zmienia ilości dostarczanego światła. Ponieważ zdolność zmiany rozmiarów soczewki jest ograniczona, jedynie przy małych odległościach możliwe jest wykorzystanie akomodacji do określania odległości.

2. Przestrzeń na obrazach ruchomych:
   - Głębia kinetyczna, paralaksa ruchu – na podstawie wzajemnego ruchu różnych elementów obrazu możliwe jest określenie wzajemnych relacji między obiektami. Zjawisko głębi kinetycznej można zaobserwować podczas jazdy samochodem lub pociągiem – obrazy przydrożnych drzew przesuwają się z dużą szybkością, natomiast rosnących dalej od drogi „przesuwają" się wolniej. Efekt ten jest powszechnie wykorzystywany w filmach, aby pokazać „przestrzeń" w której dzieje się akcja, przesuwa się kamerę.
   - Rozmycie obrazu – obiekty poruszające się z dużą szybkością postrzegane są jako „rozmazane". Na podstawie kierunku oraz wielkości rozmycia możliwa jest odtworzenie przestrzennych relacji między elementami obrazu.
3. Postrzeganie obuoczne:
   - Konwergencja oczu – dzięki niezależnemu sterowaniu obydwu gałek ocznych poprzez skupienie wzroku na wybranym obiekcie, możliwe jest śledzenie jego położenia w przestrzeni.
   - Głębokość stereowizji – obraz postrzegany przez obydwoje oczu jest trochę inny, ze względu na odległość między oczami oraz odległość do punktu, w którym skupiony jest wzrok. Na podstawie tych różnić mózg jest w stanie odtworzyć przestrzenne zależności w obserwowanym obrazie.

Umiejętne łączenie powyższych cech obrazu umożliwia uzyskanie „trójwymiarowości" nawet na płaskich, dwuwymiarowych wynikach wizualizacji drukowanych czy też wyświetlanych na ekranach. To dzięki naturalnej zdolności człowieka do interpretacji takich elementów obrazu jak wzajemne przysłaniania, zbieżność perspektywy, paralaksy ruchu czy oświetlenie, możliwe jest generowanie przez karty graficzne poprawnie interpretowanych, „syntetycznych obrazów" wirtualnych, trójwymiarowych obiektów.

# Rozdział 2
# Cyfrowa reprezentacja danych oraz popularne standardy zapisu danych

W rozdziale tym przedstawione zostaną podstawy cyfrowej reprezentacja danych oraz popularne standardy zapisu informacji takie jak prosty standard CSV, samoopisujące się pliki XML, medyczny format danych DICOM oraz uniwersalny, hierarchiczny format danych HDF.

## 2.1. Typy danych

Pojęcie *dane* można rozpatrywać w dwóch aspektach jako: *wartości* oraz *pojęcia,* kryjące się pod tymi wartościami. Wartości stanowią opis niskiego poziomu, zaś pojęcia przenoszą liczby do konstrukcji wyższego poziomu. Przykładowo, zbiór liczb {–15,3; 68,7; 129,9} można potraktować jedynie jako zbiór liczb rzeczywistych. Zbiór ten uzupełniony o *model koncepcyjny* w postaci informacji, że jest to zapis zmian temperatury, umożliwia przypisanie wartościom liczbowym informacji wyższego poziomu o różnym charakterze: porządkowym (zimno, ciepło, gorąco), nominalnym (bezpieczne, niebezpieczne) lub ilościowym (–15,3°C).

Informacje, ze względu na zastosowaną skalę, można podzielić na trzy główne *typy informacji*:

1. Dane **ilościowe** (ang. *quantitative*), w których można wyróżnić dwie grupy:
   – dane typu *interwałowego* (np. daty) mające jedynie operator różnicy umożliwiający porównywanie wielkości;
   – dane typu *ilorazowego* (np. dane pomiarowe masy, temperatury), dla których można wyznaczyć stosunek lub proporcję.
2. Dane **porządkowe** (ang. *ordinal*, *ordered*), np.: mały, średni, duży. Dane tego typu mają ściśle określone wartości oraz porządek, możliwe jest więc określenie relacji porządku ($\leqslant$, $<$, $>$, $\geqslant$). Nie można jednak obliczyć odległości między ich wartościami!
3. Dane typu **nominalnego**, kategorie, (ang. *nominal*) np.: płeć, miejsce zamieszkania. Służą głównie do „nazywania" danych, np. w postaci tekstu (jabłko, gruszka) jak też wartości liczbowych (np. numer linii tramwajowej).

Uwzględniając podstawowe typy danych występujące w systemach informatycznych, można wyróżnić wymienione dalej *modele danych.*

- **Kategoria danych** – są to dane typu nominalnego, nazywające dane, przechowywane w postaci ciągu znaków alfanumerycznych lub liczby (np. ID w systemach bazodanowych, nagłówki w tabelach opisujące wiersz lub kolumnę). Do przechowywania tego typu danych wykorzystuje się znaki alfanumeryczne, kodowane w jednym ze standardów. Znaki alfanumeryczne obejmują litery, cyfry, znaki przestankowe i sterujące oraz inne symbole. Do najpopularniejszych kodów należą 7- lub 8-bitowy kod ASCII oraz uniwersalny kod UNICODE[1] mogący występować w wersjach od 8-bitowej do 32-bitowej. Ograniczona pojemność kodu ASCII doprowadziła do powstania wielu zestawów znaków, dla różnych części świata (np. zestaw polskich liter zawierających znaki diakrytyczne). Powstało kilkanaście odmian kodowania ASCII dla języka polskiego, najpopularniejsze to IS08859-2 oraz Microsoft CP 1250. Niestety, nie są one ze sobą kompatybilne – np. polskie znaki diakrytyczne znajdują się pod różnymi kodami. Wady tej nie posiada kodowanie UNICODE, które ze względu na zastosowane dłuższe słowo, umożliwia jednoznaczne zakodowanie dowolnych znaków występujących na świecie.
- **Dane całkowitoliczbowe** – zdyskretyzowane dane ilościowe, ilorazowe obejmujące skończony zbiór liczb naturalnych. Zazwyczaj dane zapisywane są w dwójkowym systemie liczbowym, w którym ilość bitów (od 8 do 64) określa zakres zbioru. Spotyka się również zapis w kodzie BCD oraz Graya.
- Dane niecałkowitoliczbowe, **liczby rzeczywiste** – informacje obejmujące zbiór liczb rzeczywistych, umożliwiające przybliżone zapisanie np. proporcji lub liczb niewymiarnych. Do kodowania tego typu danych używa się zazwyczaj liczb zmiennoprzecinkowych, zapisywanych w postaci wykładniczej (znak, wykładnik i mantysa). Najczęściej wykorzystuje się dwie długości słowa: 32-bitowy typ danych *float* oraz 64-bitowy *double.* Istnieje również skrócona, 16-bitowa wersja zapisu zmiennoprzecinkowego, tzw. *half--float,* używana w jednostkach akceleracji obliczeń SSE5 procesorów CPU oraz w interfejsie OpenGL.

Zaprezentowane powyżej podstawowe modele danych stanowią podstawę do tworzenia bardziej skomplikowanych koncepcji, opisujących np. obiekty rzeczywiste. Przykładem bardziej skomplikowanego modelu danych może być model medycznych danych pacjenta. Mógłby on obejmować zarówno informację typu nominalnego, zapisane w postaci ciągów znaków (imię, nazwisko, adres, PESEL), dane ilościowe interwałowe (datę urodzenia i badania w postaci liczb całkowitych), informacje ilościowe ilorazowe (np. wyniki badań: ciśnienie krwi, temperatura) czy też klasyfikację przypadku (przypadek „ciężki", wymagający natychmiastowej pomocy). Poza informacjami ściśle medycznymi w systemach informatycznych, powszechnie stosuje się informacje dodatkowe, wykorzystywane do identyfikacji i organizacji danych. Klasycznym przykładem są np. sumy kontrolne, służące kontroli integralności danych czy też unikatowe identyfikatory lub wartości kluczowe. Przykładem unikatowego klucza, identyfikującego konkretną osobę, jest jej numer PESEL.

[1] http://unicode.org/

## 2.2. Formaty danych

Format zapisu danych zawiera reguły określające strukturę wewnętrzną plików, sposób rozmieszczenia, zakodowania oraz zapisu informacji. Istnieje bardzo dużo różnych formatów zapisu danych do pliku. Warto jednak zapoznać się przynajmniej z kilkoma, które można nazwać „standardami". Są wśród nich format CSV, prosty w budowie a jednocześnie o dużych możliwościach, format XML, stworzony na potrzeby Internetu i z powodzeniem stosowany do zapisu danych przez bibliotekę VTK, standard DICOM będący podstawą graficznej wymiany danych w systemach medycznych oraz uniwersalny hierarchiczny format danych HDF.

## 2.3. CSV

Format CSV (ang. *Comma Separated Values* – wartości rozdzielone przecinkiem) jest popularnym formatem przechowywania danych w plikach tekstowych, w których poszczególne wartości rozdzielane są znakiem separatora, najczęściej przecinkiem. Format ten powstał w latach 60. dwudziestego wieku na potrzeby aplikacji bazodanowych. Podstawową jednostką informacji jest w nich *rekord danych,* składający się z *pól.* Pola natomiast mogą zawierać dowolny typ danych: liczby, daty lub tekst.

Pliki w standardzie CSV, ze względu na proste zasady tworzenia oraz powszechne stosowanie od wielu lat, stały się *standardem de-facto* opisanym w dokumencie RFC 4180[2].

Istnieje bardzo wiele implementacji standardu CSV, jednak większość przestrzega następujących zasad:

- Poszczególne rekordy rozdzielane są znakiem końca wiersza CRLF, który składa się z dwóch kodów: *powrót karetki* CR (wartość dziesiętna kodu ASCII równa 13, '\r') oraz znaku *nowa linia* LF (wartość ASCII 10, '\n').
- Wartości pól rozdzielane są separatorem, zazwyczaj przecinkiem. Separatorem może być również średnik lub znak tabulacji. W jednym pliku może występować tylko jeden rodzaj separatora. Zastosowanie przecinka jest szczególnie kłopotliwe w przypadku polskich ustawień regionalnych systemu operacyjnego, ponieważ przecinek oddziela zwykle część całkowitą od ułamkowej.
- Wartości pól mogą być objęte cudzysłowem.
- Wartości pól muszą być objęte cudzysłowem, gdy zawierają znak separatora lub przejścia do nowego wiersza.
- Wartość pola może zawierać cudzysłów, pod warunkiem, że będzie on zdublowany.
- Spacje oraz inne białe znaki należą do pól.
- Pierwszy wiersz może stanowić nagłówek zawierający nazwy pól rekordów.
- Ostatni wiersz w pliku nie musi zawierać znaku CRLF.

[2] RFC 4180: Common Format and MIME Type for Comma-Separated Values (CSV) Files – dokumenty RCF (ang. *Request for Comments*) stanowią zbiór technicznych oraz organizacyjnych dokumentów związanych z Internetem i sieciami komputerowymi.

Listing 2.1 zawiera przykład pliku z danymi CSV. W pierwszym wierszu znajdują się objęte cudzysłowem identyfikatory kolumn. W kolejnych wierszach znajdują się zmiennoprzecinkowe dane liczbowe, oddzielone średnikami.

**Listing 2.1:** Przykład zawartości pliku CSV

```
1  "T";"X";"Y";"Z"
2  1000;464,0769;244,9375;1,9797
3  1500;200,6116;134,6656;7,2411
4  2000;142,0168;111,4610;16,1708
5  2500;119,3486;103,5225;27,3300
6  3000;108,4451;100,3081;39,4256
7  4000;99,4343;98,4905;63,4060
```

Format CSV może stanowić pomost podczas konwertowania danych między różnymi formatami danych, ponieważ wspierany jest przez większość programów biurowych (Microsoft Office, OpenOffice), naukowych (Matlab) oraz posiada silne wsparcie w językach skryptowych (Python, Php, Perl, Ruby). Standard ten jest również stosowany w oprogramowaniu wbudowanym w sprzęt np. w oscyloskopach czy też sprzęcie medycznym.

Prosta budowa formatu plików CSV stanowi jednocześnie poważną wadę, ponieważ w plikach tych można przechowywać praktycznie wyłącznie dane o prostych strukturach typu tablice.

## 2.4. Formaty biblioteki VTK

Standard XML powstał na potrzeby Internetu i jest językiem formalnym do reprezentowania danych w sposób strukturalny. Również ten format zapisu danych opiera się na plikach tekstowych. Poza właściwymi danymi plik XML zawiera znaczniki, tzw. *tagi,* służące do opisu zawartości pliku. Zbiory tagów tworzą tzw. *języki znaczników.* Zbiory te mogą być ściśle określone, jak ma to miejsce w przypadku wektorowego formatu zapisu grafiki SVG[3]. Biblioteka VTK posługuje się własnym językiem znaczników.

Biblioteka VTK wykorzystuje dwa rodzaje plików do przechowywania danych: pliki w standardzie XML oraz starszy format (ang. *Legacy Format*) oparty na plikach tekstowych. Szczegółowy opis obydwu standardów znajduje się pod adresem `http://vtk.org/VTK/img/file-formats.pdf`.

### 2.4.1. VTK Legacy

Plik w formacie Legacy jest plikiem tekstowym zbudowanym z następujących części:

1. Pierwsza część składa się z jednego wiersza będącego identyfikatorem: `# VTK DataFile Version xx`. Wiersz ten wygląda zawsze identycznie, różni się tylko numerem wersji `xx`.

[3] Scalable Vector Graphics (SVG) `http://www.w3.org/Graphics/SVG/`

2. Drugą cześć stanowi nagłówek o maksymalnej długości 256 znaków, zakończony znakiem końca wiersza (’/n’). Nagłówek może być wykorzystany w dowolny sposób, np. może opisywać lub identyfikować zawartość pliku.
3. Trzecią częścią jest określenie formatu danych: `ASCII` lub `BINARY`.
4. Kolejna część określa strukturę danych: geometrię i topologię zbioru danych. Ta część zaczyna się od wiersza zawierającego słowo kluczowe `DataSet`, a następnie słowo kluczowe opisujące typ danych (podrozdz. 3.2.9.):
   - `STRUCTURED_POINTS` – regularny zbiór punktów,
   - `STRUCTURED_GRID` – regularna siatka o zadanych współrzędnych,
   - `UNSTRUCTURED_GRID` – dowolne dane geometryczne i topologiczne,
   - `POLYDATA` – dane graficzne (np. siatka 3D),
   - `RECTILINEAR_GRID` – regularna siatka o zmiennych odległościach między wierszami i kolumnami,
   - `FIELD` – dowolne dane.

   Następnie, w zależności od rodzaju zbioru danych (typu danych), kolejne słowa kluczowe określają niezbędne parametry danych.
5. Ostatnia część zawiera zbiór danych. Ta część zaczyna się od słów kluczowych `POINT_DATA` lub `CELL_DATA`, po którym znajduje się liczba całkowita określająca liczbę punktów lub komórek.

Listing 2.2 zawiera przykładową zawartość pliku. Zawiera on trójwymiarowe dane wolumetryczne o rozmiarach 3 na 4 na 6, zapisane w formacie ASCII w postaci danych typu `char`.

**Listing 2.2:** Przykład zawartości pliku zapisanego w formacie VTK Legacy

```
1  # vtk DataFile Version 2.0
2  Volume example
3  ASCII
4  DATASET STRUCTURED_POINTS
5  DIMENSIONS 3 4 6
6  ASPECT_RATIO 1 1 1
7  ORIGIN 0 0 0
8  POINT_DATA 72
9  SCALARS volume_scalars char 1
10 LOOKUP_TABLE default
11 0 0 0 0 0 0 0 0 0 0 0 0
12 0 5 10 15 20 25 25 20 15 10 5 0
13 0 10 20 30 40 50 50 40 30 20 10 0
14 0 10 20 30 40 50 50 40 30 20 10 0
15 0 5 10 15 20 25 25 20 15 10 5 0
16 0 0 0 0 0 0 0 0 0 0 0 0
```

Pierwszy wiersz zawiera identyfikator, drugi opisuje zawartość, trzeci określa format danych, zaś czwarty typ danych. W wierszach 5–10 znajdują się informacje wymagane dla typu `STRUCTURED_POINTS` (regularnego zbioru punktów, szczegóły w rozdziale 3.2.5). Od wiersza 11 do końca pliku znajdują się właściwe dane liczbowe. Dokładniejszy opis biblioteki znajduje się w kolejnych rozdziałach książki, zwłaszcza w rozdziale 3.

## 2.4.2. VTK XML

Budowa plików VTK zgodnych ze standardem XML jest znacznie bardziej skomplikowana niż w przypadku VTK Legacy. Głównym powodem opracowania plików VTK zgodnych z XML było ułatwienie dostępu do danych strumieniowych i przetwarzanych równolegle. Dodatkowo opracowano wsparcie dla: kompresji danych, przenośnego kodowania binarnego (uwzględniającego architekturę 64-bitową), dostępu do danych w trybie swobodnym (ang. *random access*), wsparcia dla kolejności bajtów typu *big endian* i *little endian,* reprezentacji zbioru danych w postaci wielu plików oraz nowych rozszerzeń plików dla różnych wersji VTK. Wykorzystany standard XML dostarcza wielu udogodnień, np. możliwość rozszerzania formatu pliku z zastosowaniem nowych tagów oraz szerokiego wsparcia od strony informatycznych narzędzi, algorytmów i bibliotek do analizy składni zawartości plików XML.

Istnieją dwa główne rodzaje plików danych VTK XML:

1) Szeregowe, w którym dane przechowywane są w jednym pliku, a dostęp do niego możliwy jest w ustalonej chwili tylko dla jednego procesu.
2) Równoległe, umożliwiające zapis i odczyt danych w jednym czasie przez wiele aplikacji lub procesów współbieżnych. Dane dzielone są na części, które przypisywane są do poszczególnych procesów. Równoległe typy plików w rzeczywistości nie zawierają żadnych danych, a jedynie informacje o strukturze danych i odniesienia do „plików szeregowych", które zawierają faktyczne dane dla każdej części.

Dodatkowo wyróżnia się podział na pliki zawierające dane o regularnej (ang. *structured*) oraz nieregularnej (ang. *unstructured*) topologii.

Listing 2.3 zawiera przykład danych z listingu 2.2, zapisanych tym razem w pliku VTK XML w wersji szeregowej.

**Listing 2.3:** Przykład zawartości pliku VTK XML

```
<VTKFile type="ImageData" version="0.1" byte_order="LittleEndian" >
  <ImageData WholeExtent="0 2 0 3 0 5" Origin="0 0 0" Spacing="1 1 1">
  <Piece Extent="0 2 0 3 0 5">
    <PointData Scalars="volume_scalars">
      <DataArray type="Int8" Name="volume_scalars" format="ascii" RangeMin="0" RangeMax="50">
        0 0 0 0 0 0
        0 0 0 0 0 0
        0 5 10 15 20 25
        25 20 15 10 5 0
        0 10 20 30 40 50
        50 40 30 20 10 0
        0 10 20 30 40 50
        50 40 30 20 10 0
        0 5 10 15 20 25
        25 20 15 10 5 0
        0 0 0 0 0 0
        0 0 0 0 0 0
      </DataArray>
    </PointData>
    <CellData>
```

```
21       </CellData>
22     </Piece>
23     </ImageData>
24   </VTKFile>
```

Każdy plik VTK XML rozpoczyna się nagłówkiem zawierającym znacznik `VTKFile` kończy zaś znacznikiem zamykającym `/VTKFile`. Nagłówek zawiera informacje o typie danych, wersji i zastosowanym kodowaniu. Następnie znajdują się znaczniki definiujące niezbędne parametry typu danych, w tym przypadku dotyczą one `ImageData`. Więcej informacji o dostępnych typach danych wykorzystywanych przez bibliotekę VTK, znajduje się w podrozdz. 3.2.9.

# 2.5. DICOM

Wraz z rozwojem tomografii komputerowej, w latach 70., konieczne stało się opracowanie standardu umożliwiającego wymianę informacji graficznych oraz towarzyszących im danych medycznych między różnymi urządzeniami i systemami informatycznymi. Pierwszy zarys standardu został opracowany w 1983 roku przez American College of Radiology (ACR) oraz National Electrical Manufacturers Association (NEMA) i został opublikowany pod nazwą *ACR-NEMA Standards Publication No. 300-1985*. W 1988 roku powstała druga wersja dokumentu, zaś w 1993 roku wersja trzecia, znacznie rozbudowana i uzupełniona o nowe możliwości. Zmieniono wówczas nazwę standardu na *Digital Imaging and Communications in Medicine* – DICOM.

Standard DICOM powstał w celu popularyzacji cyfrowej wymiany danych, ułatwienia tworzenia oraz rozbudowy systemów archiwizacji obrazów PACS (ang. *Picture Archiving and Comunication Systems*) oraz wymiany informacji medycznych z innymi systemami informatycznymi stosowanymi w medycynie (ang. *Hospital Information System – HIS*). DICOM jest uzupełnieniem standardu HL7 *Health Level Seven* o zasady komunikacji i wymianę obrazów w medycynie, które to nie występują w normie HL7[4].

## 2.5.1. Dokumentacja standardu DICOM

Podstawowym źródłem informacji o standardzie DICOM jest rozbudowana dokumentacja, podzielona na kilkanaście części. Dokumentacja opisująca DICOM jest redagowana przez *NEMA Diagnostic Imaging and Therapy Systems Division* i dostępna w postaci zestawu następujących dokumentów[5]:

PS 3.1: *Introduction and Overview* – zawiera wprowadzenie oraz podstawowe informacje na temat standardu.

PS 3.2: *Conformance* – zawiera definicje podstawowych zasad oraz pojęć.

[4] Podstawowe informacje o HL7 można znaleźć na stronie `http://www.hl7.com.au/FAQ.htm`.

[5] Dokumentacja znajduje się pod adresem: `ftp://medical.nema.org/medical/dicom/2009/`

PS 3.3: *Information Object Definitions* – definiuje informatyczną reprezentację fizycznych danych – format danych w postaci zbiorów parametrów (ang. *Data Elements*).

PS 3.4: *Service Class Specifications* – definiuje metody służące do obsługi informatycznych reprezentacji danych.

PS 3.5: *Data Structure and Encoding* – określa sposób kodowania oraz niezbędne struktury wymagane do budowy pełnej informacji o rzeczywistych danych medycznych.

PS 3.6: *Data Dictionary* – zawiera tzw. *słownik danych,* czyli spis wszystkich możliwych informacji występujących w standardzie wraz z ich nazwą oraz kodami liczbowymi umożliwiającymi identyfikację danych.

PS 3.7: *Message Exchange* – definiuje sposób wymiany informacji między aplikacjami.

PS 3.8: *Network Communication Support for Message Exchange* – dokument definiuje sposób wymiany informacji między aplikacjami DICOM z wykorzystaniem protokołu TCP/IP. Standard DICOM wykorzystuje warstwową strukturę modelu ISO OSI.

PS 3.10: *Media Storage and File Format for Data Interchange* – opisuje model archiwizacji danych na nośnikach zewnętrznych. Dokument ten był podstawowym źródłem informacji na temat struktury zawartych na płycie CD danych z tomografii komputerowej.

PS 3.11: *Media Storage Application Profiles* – określa wymagania stawiane aplikacjom gromadzącym dane w standardzie DICOM.

PS 3.12: *Storage Functions and Media Formats for Data Interchange* – definiuje funkcje, nośniki oraz sposoby wymiany danych.

PS 3.14: *Grayscale Standard Display Function* – określa sposoby prezentacji danych graficznych, np. sposoby kalibracji monitorów.

PS 3.15: *Security Profiles* – opisuje aspekty bezpieczeństwa danych.

PS 3.16: *Content Mapping Resource* – definiuje szablony obiektów używanych w standardzie.

PS 3.17: *Explanatory Information* – zawiera informacje dodatkowe, uzupełniające i wyjaśniające, np. objaśnienie nomenklatury stosowanej do opisu orientacji przestrzennej pacjenta (dodatek A).

PS 3.18: *Web Access to DICOM Persistent Objects (WADO*) – opisuje uproszczony interfejs umożliwiający dostęp do części danych pacjenta poprzez pliki HTML lub XML oraz protokół HTTP/HTTPs.

Omawiany standard nie jest zamknięty. Cały czas trwają prace nad jego rozwojem, czego dowodem są liczne dodatki oraz uaktualnienia. Ze względu na stopień skomplikowania standardu, w niniejszej pracy przedstawione zostaną jedynie informacje podstawowe, umożliwiające zrozumienie jego idei oraz ułatwiające dalsze poszerzanie wiedzy w tym zakresie.

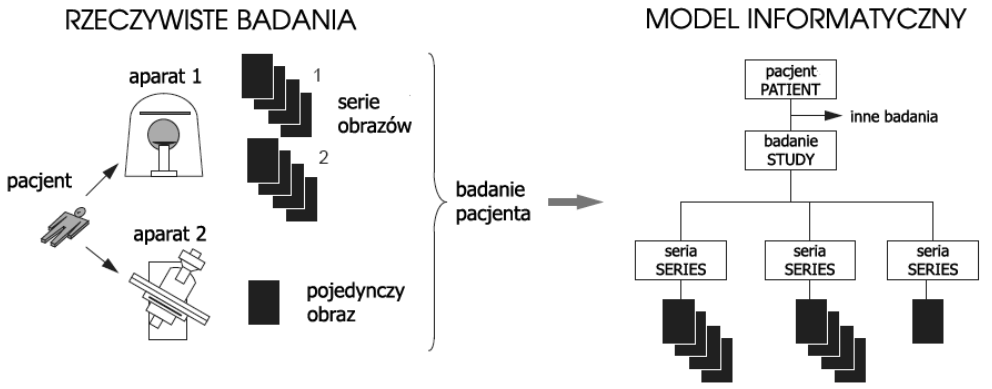
DICOM został przystosowany do sieciowej wymiany informacji (on-line) w architekturze klient-serwer, poprzez protokół TCP/IP, oraz do współpracy z nośnikami wymiennymi

(off-line), takimi jak: dyskietki, dyski CD, DVD czy MOD[6]. Wymiana danych na tych nośnikach odbywa się poprzez systemy plików ISO 9660 oraz FAT16. Dodatkowo, dla nośników fizycznych tworzony jest plik DICOMDIR przechowujący informację o wszystkich plikach DICOM na nośniku. Dość często można jednak spotkać nośniki CD lub DVD niezgodne ze standardem DICOM: nie zawierające pliku DICOMDIR jednak zawierające pliki zapisane w standardzie DICOM.

Dla metod wymiany danych zdefiniowano wspólną reprezentację danych (format danych, ang. *information object*, dokument PS 3.3) oraz rodzaje usług-serwisów (ang. *service classes*, PS 3.4), opisujących sposób współdziałania aplikacji. Współpraca aplikacji polega na wymianie specyficznych rozkazów (ang. *DICOM Commands* – PS 3.7). Dokładnie zdefiniowany jest poziom zgodności – wyszczególniono, jakie informacje muszą być zawarte w konkretnych przypadkach (PS 3.7, 3.8 i 3.10). Aby komunikacja była możliwa dwa połączone systemy ustalają role, jakie będą podczas niej pełnić. Określają, który system jest serwerem danych (ang. *Service Class Provider* – SCP), a który klientem (ang. *Service Class User* – SCU). Określają również metodę kodowania przesyłanych danych (little endian, big endian, JPG itp.).

## 2.5.2. Model informacji

Struktura danych DICOM odzwierciedla strukturę danych medycznych. Dane powiązane są w hierarchiczne drzewa, ułatwiające interpretację oraz umożliwiające łatwą ich modyfikację. Rysunek 2.1 przedstawia sposób organizacji danych zgromadzonych podczas rzeczywistych badań.




**Rys. 2.1.** Związek między danymi rzeczywistymi oraz modelem informatycznym zastosowanym w DICOM

[6] Dyski magnetooptyczne, oznaczane MOD lub M.O. (ang. *magneto-optical disc*), mają konstrukcję zbliżoną do dyskietek, występowały w dwóch rozmiarach 3,5" oraz 5,25" i mogły pomieścić od 128 MB do 9.2 GB. Ze względu na cenę nośnika technologia ta została wyparta przez płyty CD\DVD.

Standard DICOM organizuje trzy podstawowe poziomy informacji, przedstawione na rysunku 2.1 oraz 2.2, które zawierają:

- Dane pacjenta (ang. *patient*), takie jak dane personalne, data urodzenia, itp.
- Badania (ang. *study*), na które składają się dane o pacjencie, dane na temat wizyt (ang. *visit*) oraz inne informacje opisujące badanie (ang. *study content notification*). Badania gromadzą informacje na temat: elementów składowych badań (ang. *study components*) oraz procedur (ang. *modality performed procedure steps*), a także wyniki badań (ang. *results*) w postaci raportu (ang. *raport*) oraz poprawek (ang. *amendment*). Każde badanie zawiera przynajmniej jedną serię danych.
- serie danych (ang. *series*) służą do gromadzenia informacji uzyskanych podczas badania. Do informacji tych zalicza się między innymi: obrazy (ang. *image*), dane nieprzetworzone (ang. *raw data*), bitową płaszczyznę notatek (ang. *overlay*), tablicę kolorów (ang. *lookup table*) oraz krzywe opisane ciągiem punktów (ang. *curve*). Przykładem serii danych jest zestaw danych (slajdów) przedstawiających przekroje przez ciało pacjenta, otrzymane podczas rekonstrukcji danych z tomografii komputerowej CT dla konkretnych parametrów rekonstrukcji (np. rozdzielczość, odległość między przekrojami, filtr rekonstrukcji, czy parametry okna).

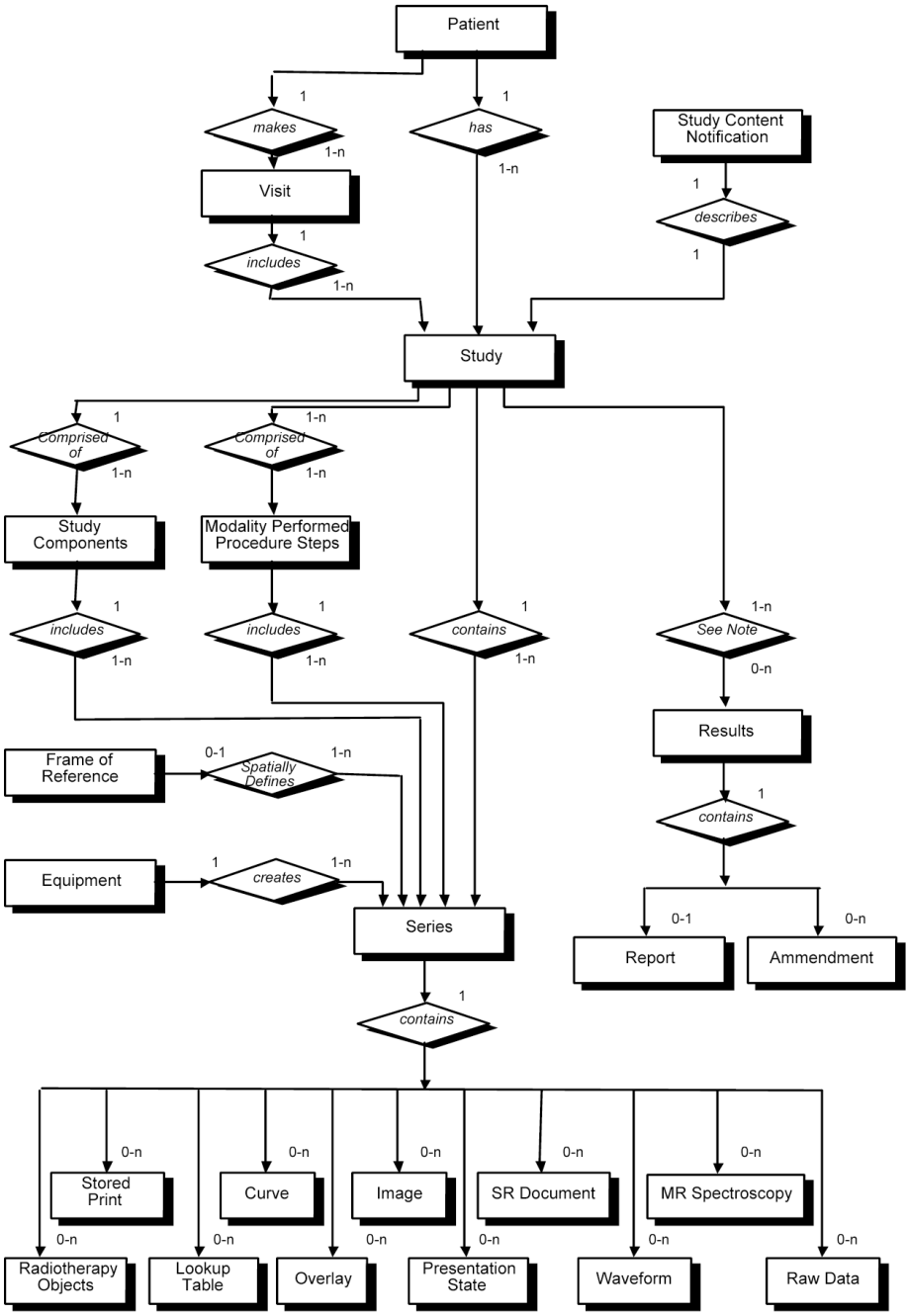
Szczegółowy model rzeczywistych danych zaimplementowany w standardzie DICOM przedstawiono na rysunku 2.2. Schemat ten pochodzi ze starszej wersji dokumentacji (2001), ze względu na mniejsze skomplikowanie i tym samym lepszą czytelność w porównaniu z najnowszą dostępną wersją dokumentacji z 2009 roku. W najnowszej dokumentacji schemat ten podzielono na kilka części (dokument PS.3: rysunki 7-la, 7-lb i 7-3).

Do standardu DICOM wprowadzono jasno sprecyzowany i dokładnie określony model informacji (ang. *Information Objects Definition* IOD, dokument PS 3.3), określający format danych dla różnych typów informacji, takich jak: obrazy, przebiegi czasowe, obiekty graficzne, raporty, wydruki itp. Model informacji IOD grupuje dane w tematycznych zbiorach, zwanych *jednostkami* (ang. *entities*) oraz podzbiorach zwanych *modułami* (ang. *module*). Każdy moduł tworzony jest przez zbiór atrybutów.

**Podstawowa jednostka danych – Data Element**

Atrybut stanowi podstawową jednostkę danych. Jest on opisywany formatem elementu danych tzw. *Data Element*, zdefiniowanym w słowniku danych (PS 3.6). *Data Element* składa się z:

- identykatora elementu danych (ang. *tag*) złożonego z dwóch liczb określających: grupę (ang. *group*) oraz element grupy (ang. *element*), zapisywanych w postaci liczb heksadecymalnych,
- typu danych (ang. *Value Representation*), określonego w postaci pary liter w kodzie ASCII i umożliwiającego poprawną interpretację danych,
- rozmiaru elementu (ang. *Value Length*) wyrażonego w bajtach,
- właściwych informacji (ang. *data*), takich jak: nazwisko pacjenta, rozdzielczość obrazu itp.




**Rys. 2.2.** Model rzeczywistych danych medycznych zaimplementowany w standardzie DICOM. zaczerpnięty z dokumentu PS 3.3

Tabela 2.1 zawiera typy danych stosowane w plikach DICOM, wraz z kodem (kolumna VR), rozwinięciem kodu oraz krótkim opisem.

**Tabela 2.1**

Kody *typów danych* VR (ang. *Value Representation*) stosowane w standardzie DICOM

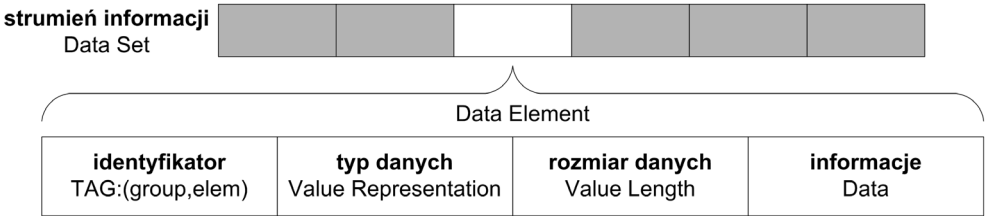
| VR | Rozwinięcie symbolu | Opis |
|---|---|---|
| AS | Age String | wiek wyrażony w dniach (nnnD), tygodniach (nnnW), miesiącach (nnnM) lub latach (nnnY), np. 018M oznacza osiemnaście miesięcy. |
| AT | Attribute Tag | para 16-bitowych liczb całkowitych bez znaku, kodujących identyfikator elementu danych: numer grupy oraz numer elementu grupy |
| CS | Code String | ciąg maksymalnie 16 znaków w kodzie ASCII |
| DA | Data | data w formacie `rrrrmmdd` (rok, miesiąc, dzień) w kodzie ASCII |
| DS | Decimal String | ciąg znaków ASCII reprezentujący liczbę zmiennoprzecinkową |
| DT | Date Time | ciąg znaków ASCII zawierający datę oraz czas w formacie: `rrrrmmddgmmss.ffffff#hhmm`, gdzie `ffffff` to ułamkowe części sekundy. `#` jest, znakiem `+/-`, zaś `hhmm` jest, offsetem godzin i minut |
| FL | Floating Point Single | 32-bitowa binarna liczba zmiennoprzecinkowa pojedynczej precyzji, zdefiniowana w dokumencie IEEE 754:1985 |
| FD | Floating Point Double | 64-bitowa binarna liczba zmiennoprzecinkowa podwójnej precyzji, zdefiniowana w dokumencie IEEE 754:1985 |
| IS | Integer String | ciąg znaków ASCII reprezentujący liczbę całkowitą ze znakiem |
| LO | Long String | ciąg maksymalnie 64 znaków, bez znaków kontrolnych |
| LT | Long Text | ciąg maksymalnie 10240 znaków ze znakami kontrolnymi |
| OB | Other Byte String | ciąg bajtów zakończony znakiem NULL |
| OF | Other Float String | ciąg 32-bitowych binarnych liczb zmiennoprzecinkowych pojedynczej precyzji |
| OW | Other Word String | ciąg 16-bitowych słów binarnych |
| PN | Person Name | ciąg znaków zawierający dane personalne pacjenta. Sposób kodowania znaków opisany jest, w elemencie danych o identyfikatorze (0008,0005) |
| SH | Short String | ciąg maksymalnie 16 znaków |

**Tabela 2.1** cd.

| | | |
|---|---|---|
| SL | Signed Long | 32-bitowa liczba całkowita ze znakiem |
| SQ | Sequence of Items | sekwencja grup elementów danych (ang. *items*) |
| SS | Signed Short | 16-bitowa liczba całkowita ze znakiem |
| ST | Short Text | ciąg znaków, zawierający znaki sterujące, o maksymalnej długości 1024. Sposób kodowania znaków opisany jest w elemencie danych o identyfikatorze (0008,0005) |
| TM | Time | ciąg znaków ASCII reprezentujący godzinę w formacie: `ggmmss.ffffff`, gdzie `fffff` to milionowe części sekundy |
| UI | Unique Identifier (UID) | ciąg znaków zawierający unikatowy identyfikator UID, identyfikujący obiekty DICOM |
| UL | Unsigned Long | 32-bitowa liczba całkowita bez znaku |
| UN | Unknown | ciąg bajtów o nieznanym sposobie kodowania |
| US | Unsigned Short | 16-bitowa liczba całkowita bez znaku |
| UT | Unlimited text | ciąg znaków, zawierający znaki sterujące, przechowujący jeden lub wiele akapitów tekstu. Sposób kodowania znaków opisany jest, w elemencie danych o identyfikatorze (0008,0005). |

### Strumień informacji – Data Set

Strumień informacyjny (ang. *Data Set*) jest uporządkowanym strumieniem elementów danych (ang. *Data Element*), co przedstawiono na rysunku 2.3. Norma DICOM, w dokumencie PS 3.3, w pełni opisuje katalog typów danych, rodzaju danych, itp.



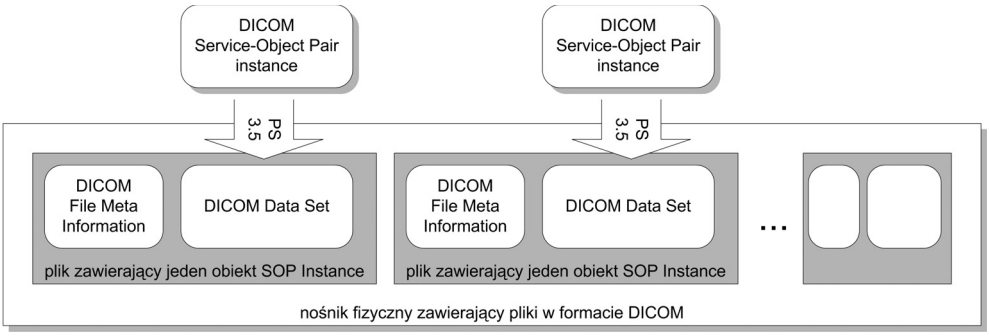

**Rys. 2.3.** Budowa strumienia informacji DICOM

Oprócz definicji obiektu danych istotne jest skojarzenie z nim elementu usługi, czyli tzw. *Service Element*. Obiekt *Service-Obiect Pair* (SOP) łączy dane z serwisami i definiuje różne usługi związane z modelem informacji IOD. W standardzie DICOM wprowadzono również techniki umożliwiające jednoznaczną identyfikację oraz ochronę danych (PS 3.15). Pozwalają one na sprecyzowanie zależności między danymi, np. podczas transmisji przez

sieć. Norma nie definiuje szczegółów implementacji, wymogów dotyczących możliwości i funkcji urządzeń oraz sposobu testowania czy zatwierdzania zgodności konkretnej implementacji ze standardem DICOM. Z tego też powodu zdarza się, że dwa różne urządzenia, zgodne ze standardem DICOM, nie są zdolne do wymiany informacji! Jest to przejawem walki różnych producentów o wpływy na rynku.

## 2.5.3. Dane CT zapisane na nośniku fizycznym

Dane będące wynikiem np. badania z wykorzystaniem tomografii komputerowej (CT) lub rezonansu magnetycznego (MRI) zapisywane są na płycie CD-R w standardzie DICOM. Zawartość płyty zdefiniowana jest w dokumentacji standardu w dokumencie PS 3.10. Rysunek 2.4 przedstawia schematyczną zawartość fizycznego nośnika informacji.

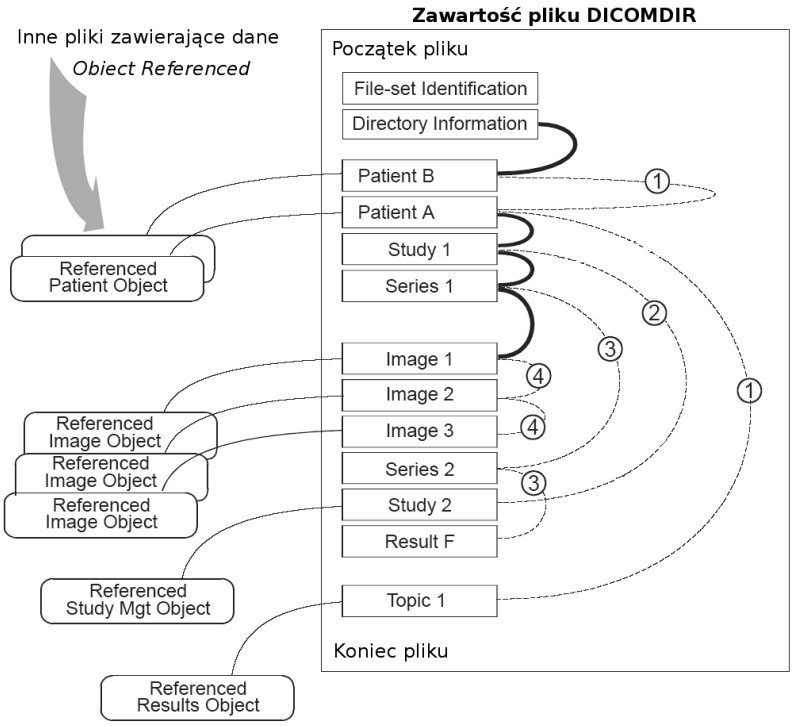



**Rys. 2.4.** Struktura danych DICOM na fizycznym nośniku danych

Fizyczny nośnik danych zawiera pliki w formacie opisanym w obiekcie *DICOM File Format.* Obiekt ten służy do przechowywania informacji w postaci plików. Każdy plik zawiera informację na temat jednego obiektu *Service-Obiect Pair instance* (SOP instance) oraz informacje o pliku tzw. *DICOM File Meta Information.*

Każdy obiekt DICOM File Meta Information zbudowany jest ze 128-bajtowej preambuły wypełnionej zerami, identyfikatora pliku DICOM (ang. *DICOM prefix*) zawierającego czteroliterowy ciąg liter „DICM" oraz zestawu elementów danych (ang. *Data Element*), opisujących plik. Elementy te mają identyfikator postaci (`0002,xxxx`). Po elementach danych należących do informacji o pliku (*DICOM File Meta Information*), znajduje się zakodowany zgodnie z dokumentem PS 3.5 ciąg elementów danych (*DICOM Data Set*), prezentujący jeden obiekt *Service-Obiect Pair instance.*

Obiektem SOP może być np. jeden przekrój CT (obraz jednej rekonstrukcji danych CT) lub obiekt opisujący dane na nośniku, tworzący plik DICOMDIR. Nośnik fizyczny zawiera katalogi z plikami, w których znajdują się informacje medyczne. Oprócz nich zawsze zapisywany jest specjalny plik o nazwie DICOMDIR opisujący zawartość nośnika. Rysunek 2.5 przedstawia budowę pliku DICOMDIR oraz sposób powiązania zawartości z innymi plikami na nośniku.




**Rys. 2.5.** Zawartość pliku DICOMDIR. PS 3.x

Plik DICOMDIR zawiera informacje na temat wszystkich plików DICOM na nośniku. Niektóre Data Elements obiektów SOP znajdują się tylko w pliku danych, a niektóre tylko w DICOMDIR. Dane w pliku DICOMDIR powiązane są w hierarchiczne drzewo, którego korzeniem jest tzw. *Directory Information.* Kolejnym poziomem są dane o pacjentach, następnym badania, potem serie, a na końcu obrazy. Przejścia między kolejnymi poziomami zaznaczone są na rysunku pogrubioną linią, zaś przejścia między obiektami na tym samym poziomie liniami z numerami poziomów. W obiektach zapisanych w DICOMDIR przechowywane są wskaźniki do danych na tym samym i kolejnym poziomie. Dzięki temu, możliwe jest szybkie przeszukiwanie danych w bardzo dużych plikach. Każdy obiekt SOP zapisany w DICOMDIR posiada również informację o pliku na nośniku (w postaci ścieżki dostępu oraz nazwy), w którym znajduje się pełna informacja o nim.

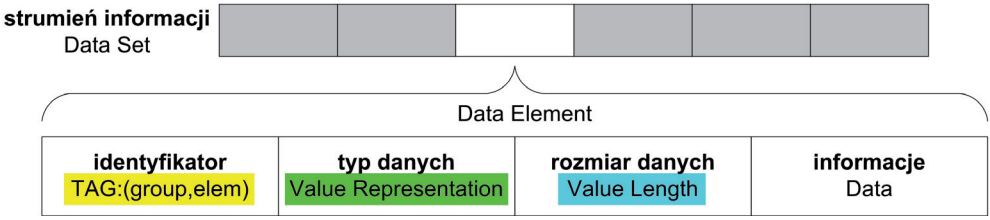
## 2.5.4. Binarna zawartość pliku DICOM

W przedstawionym na rysunku 2.6 fragmencie pliku DICOM, można wyróżnić: 128-bajtową pustą preambułę pliku, identyfikator w postaci czterech liter w kodzie ASCII „DICM", po którym następuje ciąg danych (strumień informacji – ang. *Data Set*) w postaci tzw. atrybutów – elementów danych (ang. *Data Element*). Dla łatwiejszej analizy poszcze-

gólnych fragmentów kodu DICOM, na rysunku 2.7 jeszcze raz umieszczono schemat budowy strumienia informacji. Dodatkowo na obu rysunkach (2.6 i 2.7) kolorami zaznaczono odpowiadające sobie części Data Element.

```
  adres  :          zawartość pliku hexadecymalnie          ; zawartość w ASCII

00000000h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000010h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000020h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000030h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000040h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000050h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000060h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................
00000070h: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ; ................ ↑ 128B preambuła
00000080h: 44 49 43 4D 02 00 00 00 55 4C 04 00 B2 00 00 00 ; DICM....UL..²...
00000090h: 02 00 01 00 4F 42 00 00 02 00 00 00 00 01 02 00 ; ....OB..........
000000a0h: 02 00 55 49 14 00 31 2E 32 2E 38 34 30 2E 31 30 ; ..UI..1.2.840.10
000000b0h: 30 30 38 2E 31 2E 33 2E 31 30 02 00 03 00 55 49 ; 008.1.3.10....UI
000000c0h: 30 00 31 2E 33 2E 31 32 2E 32 2E 31 31 30 37 2E ; 0.1.3.12.2.1107.
.........  .. .. .. .. .. .. .. .. .. .. .. .. .. .. ..   ................

  legenda oznaczeń: identyfikator, TAG=(group,element), VR, długość danych, dane
```

**Rys. 2.6.** Przykładowy początek pliku DICOM




**Rys. 2.7.** Budowa strumienia danych DICOM

Listing 2.4 zawiera zdekodowany fragment pliku DICOMDIR, otrzymany przy pomocy programu dcmdump w wersji 3.5.3, będącego częścią projektu *DICOM ToolKit* (dcmtk[7]). Program dcmdump pracuje w trybie tekstowym i umożliwia analizę zawartości plików DICOM. Za pomocą wcięć tekstu program zaznacza kolejne poziomy hierarchii danych.

W prezentowanym fragmencie pliku DICOMDIR usunięto część informacji o podobnej budowie, a pozostawiono jedynie bloki prezentujące podstawowe struktury informacji, takie jak: grupę opisującą pacjenta, badanie, serię czy obrazy.

**Listing 2.4:** Zdekodowana zawartość pliku DICOMDIR

```
1  # Dicom-File-Format
2
3  # Dicom-Meta-Information-Header
4  # Used TransferSyntax: LittleEndianExplicit
```

[7] Strona projektu: `http://dicom.offis.de/dcmtk.php.en`

```
(0002,0000) UL   178                                #    4,  1  MetaElementGroupLength
(0002,0001) OB   00\01                              #    2,  1  FileMetaInformationVersion
(0002,0002) UI   =BasicDirectoryStorageSOPClass     #   20,  1  MediaStorageSOPClassUID
(0002,0003) UI   [1.3.12.2.1107.5.1.4.53013.20040415180226043700O] #   48, 1
  MediaStorageSOPInstanceUID
(0002,0010) UI   =LittleEndianExplicit              #   20,  1  TransferSyntaxUID
(0002,0012) UI   [1.3.12.2.1107.5.1.4]              #   20,  1  ImplementationClassUID
(0002,0013) SH   [SIEMENS_S5VA70A]                  #   16,  1  ImplementationVersionName

# Dicom-Data-Set
# Used TransferSyntax: LittleEndianExplicit
(0004,1130) CS   [C******O]                         #    8,  1  FileSetID
(0004,1200) up   $384                               #    4,  1
  OffsetOfTheFirstDirectoryRecordOfTheRootDirectoryEntity
(0004,1202) up   $384                               #    4,  1
  OffsetOfTheLastDirectoryRecordOfTheRootDirectoryEntity
(0004,1212) US   0                                  #    2,  1  FileSetConsistencyFlag
(0004,1220) SQ   (Sequence with explicit Length #=707) # 2256576, 1  DirectoryRecordSequence
    (fffe,e000) na    "Directory Record" PATIENT #=9   #     138,  1  Item
    #   offset= $384
       (0004,1400) up   $0                          #    4,  1  OffsetOfTheNextDirectoryRecord
       (0004,1410) US   65535                       #    2,  1  RecordInUseFlag
       (0004,1420) up   $530                        #    4,  1
         OffsetOfReferencedLowerLevelDirectoryEntity
       (0004,1430) CS   [PATIENT]                   #    8,  1  DirectoryRecordType
       (0008,0005) CS   [ISO_IR 100]                #   10,  1  SpecificCharacterSet
       (0010,0010) PN   [**********^******]         #   16,  1  PatientsName
       (0010,0020) LO   [1234/56/7890]              #   12,  1  PatientID
       (0010,0030) DA   [19510101]                  #    8,  1  PatientsBirthDate
       (0010,0040) CS   [M]                         #    2,  1  PatientsSex
    (fffe,e00d) na   "ItemDelimitationItem for re-encoding" #    0,  1  ItemDelimitationItem
    (fffe,e000) na   "Directory Record" STUDY #=11          #  210,  1  Item
    #   offset= $530
       (0004,1400) up   $0                          #    4,  1  OffsetOfTheNextDirectoryRecord
       (0004,1410) US   65535                       #    2,  1  RecordInUseFlag
       (0004,1420) up   $748                        #    4,  1
         OffsetOfReferencedLowerLevelDirectoryEntity
       (0004,1430) CS   [STUDY]                     #    6,  1  DirectoryRecordType
       (0008,0005) CS   [ISO_IR 100]                #   10,  1  SpecificCharacterSet
       (0008,0020) DA   [2004****]                  #    8,  1  StudyDate
       (0008,0030) TM   [093859.265000]             #   14,  1  StudyTime
       (0008,0050) SH   (no value available)        #    0,  0  AccessionNumber
       (0008,1030) LO   [Thorax^THORAX ROUTINE +C]#   24,  1  StudyDescription
       (0020,000d) UI   [1.3.12.2.1107.5.1.4.53000.4.0.10975916227400000] #    48,  1
         StudyInstanceUID
       (0020,0010) SH   [1]                         #    2,  1  StudyID
    (fffe,e00d) na   "ItemDelimitationItem for re-encoding" #     0,  1  ItemDelimitationItem
    (fffe,e000) na   "Directory Record" SERIES #=13#            260,  1  Item
    #   offset= $748
       (0004,1400) up   $5288                       #    4,  1  OffsetOfTheNextDirectoryRecord
       (0004,1410) US   65535                       #    2,  1  RecordInUseFlag
       (0004,1420) up   $1016                       #    4,  1
         OffsetOfReferencedLowerLevelDirectoryEntity
       (0004,1430) CS   [SERIES]                    #    6,  1  DirectoryRecordType
       (0008,0005) CS   [ISO_IR 100]                #   10,  1  SpecificCharacterSet
       (0008,0021) DA   [2004****]                  #    8,  1  SeriesDate
       (0008,0031) TM   [175748.406000]             #   14,  1  SeriesTime
       (0008,0060) CS   [CT]                        #    2,  1  Modality
```

```
56        (0008,0080) LO   [Kat. i Zakl. Radiologii] #   24,  1  InstitutionName
57        (0008,0081) ST   [             ]           #   14,  1  InstitutionAddress
58        (0008,103e) LO   [<SSD Collection>]        #   16,  1  SeriesDescription
59        (0020,000e) UI   [1.3.12.2.1107.5.1.4.53013.204.0.4606256229580000] #   48,  1
            SeriesInstanceUID
60        (0020,0011) IS   [607]                     #    4,  1  SeriesNumber
61     (fffe,e00d) na   "ItemDelimitationItem for re-encoding" #      0,  1  ItemDelimitationItem
62     (fffe,e000) na   "Directory Record" IMAGE #=18 #  418,  1  Item
63     #    offset= $1016 refFileID="04151802\46013825"
64        (0004,1400) up   $1442                     #    4,  1  OffsetOfTheNextDirectoryRecord
65        (0004,1410) US   65535                     #    2,  1  RecordInUseFlag
66        (0004,1420) up   $0                        #    4,  1
            OffsetOfReferencedLowerLevelDirectoryEntity
67        (0004,1430) CS   [IMAGE]                   #    6,  1  DirectoryRecordType
68        (0004,1500) CS   [04151802\46013825]       #   18,  2  ReferencedFileID
69        (0004,1510) UI   =SecondaryCaptureImageStorage  #   26,  1  ReferencedSOPClassUIDInFile
70        (0004,1511) UI   [1.3.12.2.1107.5.1.4.53013.204.0.4606444851200000] #   48,  1
            ReferencedSOPInstanceUIDInFile
71        (0004,1512) UI   =LittleEndianExplicit     #   20,  1  ReferencedTransferSyntaxUIDInFile
72        (0008,0005) CS   [ISO_IR 100]              #   10,  1  SpecificCharacterSet
73        (0008,0008) CS   [DERIVED\SECONDARY\OTHER\CSA PSSD\]   #   34,   5  ImageType
74        (0008,0016) UI   =SecondaryCaptureImageStorage  #   26,  1  SOPClassUID
75        (0008,0018) UI   [1.3.12.2.1107.5.1.4.53013.204.0.4606444851200000] #   48,  1
            SOPInstanceUID
76        (0008,0023) DA   [2004****]                #    8,  1  ContentDate
77        (0008,0033) TM   [175837.890000]           #   14,  1  ContentTime
78        (0020,0013) IS   [61]                      #    2,  1  InstanceNumber
79        (0028,0010) US   512                       #    2,  1  Rows
80        (0028,0011) US   512                       #    2,  1  Columns
81        (0050,0004) CS   (no value available)      #    0,  0  CalibrationImage
82     (fffe,e00d) na   "ItemDelimitationItem for re-encoding" #    0,  1  ItemDelimitationItem
83
84       ...
85       ...
86       ...
87
88      (fffe,e000) na   "Directory Record" IMAGE #=24# 4884,  1  Item
89      #   offset= $2252068 refFileID="04151802\46082139"
90        (0004,1400) up   $0                        #    4,  1  OffsetOfTheNextDirectoryRecord
91        (0004,1410) US   65535                     #    2,  1  RecordInUseFlag
92        (0004,1420) up   $0                        #    4,  1
            OffsetOfReferencedLowerLevelDirectoryEntity
93        (0004,1430) CS   [IMAGE]                   #    6,  1  DirectoryRecordType
94        (0004,1500) CS   [04151802\46082139]       #   18,  2  ReferencedFileID
95        (0004,1510) UI   =CTImageStorage           #   26,  1  ReferencedSOPClassUIDInFile
96        (0004,1511) UI   [1.3.12.2.1107.5.1.4.53013.4.0.11012094322280000] #   48,  1
            ReferencedSOPInstanceUIDInFile
97        (0004,1512) UI   =LittleEndianExplicit     #   20,  1  ReferencedTransferSyntaxUIDInFile
98        (0008,0005) CS   [ISO_IR 100]              #   10,  1  SpecificCharacterSet
99        (0008,0008) CS   [ORIGINAL\PRIMARY\AXIAL\CT_SOM5 SPI]  #    34,  4  ImageType
100       (0008,0016) UI   =CTImageStorage           #   26,  1  SOPClassUID
101       (0008,0018) UI   [1.3.12.2.1107.5.1.4.53013.4.0.11012094322280000] #    48,  1
            SOPInstanceUID
102       (0008,0023) DA   [2004****]                #    8,  1  ContentDate
103       (0008,0033) TM   [094216.790425]           #   14,  1  ContentTime
104       (0008,1140) SQ   (Sequence with explicit Length #=1) #   98,  1
            ReferencedImageSequence
105           (fffe,e000) na   (Item with explicit Length #=2) #   90,  1  Item
```

```
106               (0008,1150) UI   =CTImageStorage   #   26,  1  ReferencedSOPClassUID
107               (0008,1155) UI   [1.3.12.2.1107.5.1.4.53013.4.0.11010935660010000] #    48,  1
                    ReferencedSOPInstanceUID
108            (fffe,e00d) na    (ItemDelimitationItem for re-encoding) #    0,  1
                 ItemDelimitationItem
109         (fffe,e0dd) na   (SequenceDelimitationItem for re-enc.) #    0,  1
              SequenceDelimitationItem
110         (0020,0013) IS   [40]                      #    2,  1  InstanceNumber
111         (0020,0032) DS   [-167.66797\-294.66797\-290]   #   26,  3  ImagePositionPatient
112         (0020,0037) DS   [1\0\0\0\1\0]             #   12,  6  ImageOrientationPatient
113         (0020,0052) UI   [1.3.12.2.1107.5.1.4.53013.4.0.11010931428830000] #    48,  1
              FrameOfReferenceUID
114         (0028,0010) US   512                       #    2,  1  Rows
115         (0028,0011) US   512                       #    2,  1  Columns
116         (0028,0030) DS   [0.6640625\0.6640625]     #   20,  2  PixelSpacing
117         (0050,0004) CS   (no value available)      #    0,  0  CalibrationImage
118         (0088,0200) SQ   (Sequence with explicit Length #=1)   #  4206,  1  IconImageSequence
119            (fffe,e000) na   (Item with explicit Length  #=9)     #  4198,  1  Item
120               (0028,0002) US   1                   #    2,  1  SamplesPerPixel
121               (0028,0004) CS   [MONOCHROME2]       #   12,  1  PhotometricInterpretation
122               (0028,0010) US   64                  #    2,  1  Rows
123               (0028,0011) US   64                  #    2,  1  Columns
124               (0028,0100) US   8                   #    2,  1  BitsAllocated
125               (0028,0101) US   8                   #    2,  1  BitsStored
126               (0028,0102) US   7                   #    2,  1  HighBit
127               (0028,0103) US   0                   #    2,  1  PixelRepresentation
128               (7fe0,0010) OW   0a0a\0a0a\0a0a\...# 4096,  1  PixelData
129            (fffe,e00d) na    (ItemDelimitationItem for re-encoding) #    0,  1
                 ItemDelimitationItem
130         (fffe,e0dd) na    (SequenceDelimitationItem for re-enc.) #    0,  1
              SequenceDelimitationItem
131      (fffe,e00d) na   "ItemDelimitationItem for re-encoding" #    0,  1  ItemDelimitationItem
132   (fffe,e0dd) na   (SequenceDelimitationItem for re-enc.) #    0,  1  SequenceDelimitationItem$
```

Każdy wiersz zawiera informację o jednym elemencie danych (ang. *Data Element*) i zbudowany jest z dwóch głównych części rozdzielonych znakiem krzyżyka (#):

- zdekodowanej informacji DICOM (trzy pierwsze wartości):
  - identyfikator elementu danych (Tag) w formacie pary heksadecymalnych liczb,
  - typ danych (VR) w postaci kodu,
  - dane lub część danych,
- informacji dodatkowych, takich jak: długość danych w bajtach, liczba podciągów danych (liczba zmiennych zapisanych w postaci znaków ASCII w bloku danych) oraz nazwa elementu danych, określona na podstawie numeru grupy i numeru elementu grupy, a zdefiniowana w dokumencie *PS 3.6: Data Dictionary.*

**Logiczna budowa pliku**

Dane zawarte w każdym pliku DICOM podzielone są na dwie części: część zawierającą informacje o pliku (*Dicom-Meta-Information-Header*) oraz dane jednego obiektu *Service--Obiect Pair Instance (Dicom-Data-Set*). Obydwie części zawierają dane w postaci strumienia informacji (ang. *Data Set*), który jest uporządkowanym strumieniem elementów danych (ang. *Data Element*). Strumień ten można zaobserwować w przytoczonym powyżej binar-

nym fragmencie pliku DICOM. Informacje o pliku (*Dicom-Meta-Information-Header*) są wymagane dla każdego pliku DICOM. Dokument PS 3.10 ściśle określa zawartość tej części pliku. Część zawierająca dane *(Dicom-Data-Set*) przechowuje informacje o jednym obiekcie typu *Service-Obiect-Pair instance* (SOP instance). Obiektem tym może być np.: pojedynczy przekrój CT, pojedyncza klatka wirtualnej bronchoskopii wygenerowana na aparacie CT lub opis zawartości nośnika, tak jak ma to miejsce w przypadku pliku DICOMDIR.

**Analiza zawartości pliku DICOMDIR**

W przytoczonym zdekodowanym fragmencie pliku DICOMDIR (listing 2.4), w części zawierającej dane (od wiersza 13), po czterech rekordach opisujących strukturę pliku (wiersze 15–18), następuje sekwencja danych określająca strukturę danych na nośniku. Zaczyna się ona od elementu danych (`0004,1220`), opisującego DirectoryRecordSequence (wiersz 19). Każda *sekwencja danych* zawiera zestawy danych nazywane *item,* zaczynające się od elementu danych o identyfikatorze (ang. *Data Element Tag*) równym (`fffe,e000`), a kończone tzw. ItemDelimitationItem, elementem danych o identyfikatorze (`fffe,e00d`). W praktyce, element określający koniec zestawu danych (ItemDelimitationItem) nie występuje ficzycznie w plikach.

W prezentowanym przykładzie, pierwszy zestawy danych (ang. *item*) przechowuje informacje o pacjencie. Zawartość poszczególnych zestawów danych definiuje element danych nazywany *DirectoryRecordType* o identyfikatorze (`0004,1430`).

W listingu 2.4 można wyróżnić fragment opisujący pacjenta, który obejmuje linie od 20 do 31. Zestaw danych (ang. *item*) (linia 20) zawiera: informacje na temat hierarchii danych (elementy danych o numerze grupy 0004), sposób kodowania danych (`SpecificCharacterSet` – (`0008,0005`)) oraz dane o pacjencie (o numerze grupy `0010`), takie jak: nazwisko pacjenta (`0010,0010`), identyfikator pacjenta (`0010,0020`), data urodzenia (`0010,0030`), płeć (`0010,0040`). Ze względu na ochronę danych osobowych, rzeczywiste informacje zostały w kodzie zastąpione wartościami przypadkowymi.

Kolejny zestaw danych (listing 2.4, wiersz 32) opisuje badanie (ang. *study*). Budowa zestawu jest podobna do informacji o pacjencie, tzn. najpierw występują informacje organizujące strukturę pliku (grupa numer `0004`), standard kodowania znaków (`0008,0005`), a następnie właściwe informacje o badaniu (grupa `0008`), takie jak: data i godzina badania, opis badania, unikatowy kod UID (ang. *Unique Identifier*) oraz numer badania (wiersze 58 i 59).

Trzeci zestaw danych (listing 2.4, wiersz 46) opisuje serię danych (ang. *series*). Również w tym przypadku występują informacje organizujące dane, a także opis konkretnej serii danych (od wiersza 53): data i czas, pochodzenie serii (modalność, dane CT), nazwa oraz adres instytucji wykonującej badanie, opis serii (wiersz 58). W tym przypadku jest to wirtualna bronchoskopia wykonana na aparacie tomografii komputerowej – SSD Collection (ang. *Shaded Surface Display*). Na końcu jest unikatowy kod UID (wiersz 59) oraz numer serii (wiersz 60).

Po opisie serii, od wiersza 62, następuje opis poszczególnych obrazów (ang. *image*). W przedstawionym w listingu 2.4 zdekodowanym fragmencie pliku DICOMDIR, znajdują się dwa zestawy danych opisujące pliki z danymi graficznymi – elementy danych `Directory-RecordType` (wiersze 67 i 93) o identyfikatorach (`0004,1430`) zawierają wpis IMAGE. Pierwszy opisuje wirtualną bronchoskopię, drugi pojedynczy przekrój tomografii kompute-

rowej. Typ przechowywanych danych określany jest w elemencie danych `ImageType` o identyfikatorze (`0008,0008`) (wiersze 73 i 99).

Jedną z ważniejszych informacji zawartych w zestawie danych jest ścieżka dostępu oraz nazwa pliku z danymi. Informacja ta zawarta jest w elemencie danych `ReferencedFilelD`, o identyfikatorze (`0004,1500`), w postaci łańcucha znaków (VR = CS, code string). W obydwu przytoczonych w listingu 2.4 przykładowych zestawach danych, występują informacje o dacie i czasie powstania obrazów, unikatowych numerach UID, numerze obrazu w serii (`InstanceNumber`) oraz rozdzielczości (ilość wierszy i kolumn obrazów).

W zestawie danych opisującym obraz z tomografii komputerowej występują dwie sekwencje danych: pierwsza zawiera `ReferencedSOPInstanceUID` jednoznacznie opisujący obiekt SOP, druga ikonę obrazu – miniaturę o niskiej rozdzielczości, `IconImageSequence`. Dodatkowo, zapisane są informacje o przestrzennych współrzędnych obrazu (`ImagePositionPatient`), orientacji obrazu w przestrzeni (`ImageOrientationPatient`) oraz odległości między pikselami (`PixelSpacing`).

### 2.5.5. Podsumowanie

Z danymi w standardzie DICOM można się coraz częściej spotkać, głownie ze względu na coraz powszechniejsze stosowanie nowoczesnych badań radiologicznych. Pracownie radiologiczne często zapisują wyniki tomografii komputerowej lub rezonansu pacjentów na płytkach CD\DVD. Również podczas pracy naukowej można spotkać się z nimi. Jednak dostęp do danych zapisanych całkowicie zgodnie ze standardem jest rzadkością. Niestety, normą jest tworzenie nośników o luźno traktowanej strukturze plików, z dołączonym oprogramowaniem do przeglądania danych i bardzo często bez pliku DICOMDIR. Przed użyciem takiego niestandardowego nośnika, należy sprawdzić zawartość wszystkich plików. Te, które zawierają dane zapisane w formacie DICOM, łącznie z wygenerowanym na ich podstawie plikiem DICOMDIR mogą być używane jako nośnik zgodny ze standardem DICOM.

Przedstawione szczegóły budowy plików DICOM miały na celu przybliżenie zasad rządzących wewnętrzną budową plików w tym standardzie. Wewnętrzna struktura plików, jak wynika z doświadczeń autora, jest zazwyczaj traktowana z mniejszą dowolnością.

Dane w standardzie DICOM są również obsługiwane przez środowisko Matlab. Listę dostępnych funkcji w konkretnej wersji oprogramowania można uzyskać komendą: `lookfor dicom`. Wersja 2008a ma następujące funkcje:

- `dicomanon` – narzędzie do usuwania danych wrażliwych (danych personalnych pacjenta) z plików DICOM,
- `dicomdict` – umożliwia obsługę tzw. *słownika danych* (dokument PS 3.6: Data Dictionary), który dla programu Matlab, zawarty jest w pliku `dicom-dict.txt`,
- `dicominfo` – umożliwia odczytanie danych niegraficznych z pliku DICOM,
- `dicomlookup` – wyszukiwanie danych w słowniku,
- `dicomread` – umożliwia wczytanie danych graficznych z pliku DICOM.
- `dicomuid` – umożliwia wygenerowanie unikatowego identyfikatora (UID),
- `dicomwrite` – zapisuje dane do pliku DICOM,
- `isdicom` – funkcja sprawdzająca, czy zadany plik może być plikiem DICOM.

Przykładowe dane DICOM można znaleźć między innymi pod następującymi adresami:

- zbiór danych udostępniony przez jednego z twórców VTK Sebastien Barre: `http://www.barre.nom.fr/medical/samples/`
- archiwum danych dla programu OsiriX: `http://pubimage.hcuge.ch:8080/`
- dane kardiologiczne: `http://www.rubomed.com/dicomfiles/index.html`

Szczególnie godna polecenia jest strona `http://www.idoimaging.com`, która zawiera bardzo wiele narzędzi informatycznych wspierających standard DICOM. Można tam również znaleźć gotowe przeglądarki, konwertery, systemy PACS (ang. *Picture Archiving and Communication System*) oraz narzędzia do tworzenia własnego oprogramowania, np. biblioteki. Godne polecenia są:

- GDCM (Grassroots DICOM library), która jest ciągle rozwijana na zasadach wolnego oprogramowania, `http://sourceforge.net/apps/mediawiki/gdcm/index.php?title=Main_Page`,
- DCMTK (DICOM Toolkit), opracowana przez twórców standardu, obecnie nie rozwijana, `http://dicom.offis.de/dcmtk`.

## 2.6. HDF

Warto również wspomnieć o jeszcze jednym sposobie przechowywania danych naukowych, a mianowicie o *hierarchicznym formacie danych HDF* (ang. *Hierarchical Data Format*), który został opracowany w National Center for Supercomputing Applications, Uniwersytet Illinois w Urbana-Champaign (USA). Obecnie, nad rozwojem standardu czuwa organizacja typu non-profit *The HDF Group*[8].

Standard HDF powstał w celu ułatwienia zarządzania bardzo dużymi zbiorami danych naukowych, takimi jak np. petabajty informacji z sond kosmicznych, danych pomiarowych lub wyników symulacji. Na standard HDF składają się specyfikacja formatu danych oraz jej implementacja dostępna w postaci biblioteki napisanej w języku C. Biblioteka umożliwia: tworzenie, analizę, przechowywanie oraz dostęp do zbiorów danych o skomplikowanej strukturze. Dane mogą być dowolnego typu, dostępne są liczne typy wbudowane, jak również istnieje możliwość tworzenia dowolnych własnych. Informacje mogą być dzielone w grupy, mogą również zawierać złożone relacje czy też inne zależności. Format HDF jest obecnie obsługiwany przez wiele komercyjnych oraz niekomercyjnych programów i języków, takich jak np. Matlab, IDL, GDL, C++, Java, Python czy Fortran.

Hierarchiczny format danych HDF dostępny jest dla wielu platform sprzętowych oraz systemów operacyjnych w postaci pakietów (dystrybucji), które zawierają zazwyczaj skompilowaną bibliotekę obsługującą pliki HDF, programy narzędziowe ułatwiające obsługę plików HDF z poziomu linii komend oraz przeglądarkę plików HDFView zaimplementowaną w Javie. Dostępne są również kody źródłowe biblioteki. Tworzone pliki HDF są niezależne od platformy sprzętowo-programowej, co bardzo ułatwia wymianę danych między różnymi systemami.

[8] `http://www.hdfgroup.org`

Istnieją dwie główne implementacje standardu HDF: starcza wersja (pierwsza wersja powstała w 1988 roku) oznaczana numerem cztery HDF4 oraz nowsza HDF5. Dodatkowo, ze względu na kompatybilność, wspierane są obecnie dwie gałęzie biblioteki HDF5 o numerach 1.8 oraz 1.6. Główną różnicą między wersjami HDF4 i HDF5 jest przebudowanie modelu danych w nowszej wersji do postaci bardzo uniwersalnych dwóch typów obiektów: wielowymiarowej tablicy danych oraz struktury tworzącej grupę obiektów. Takie podejście zapewnia bardzo dużą elastyczność, jednak znacznie komplikuje nawet podstawowe operacje. Starsza wersja (HDF4) zawiera aż osiem podstawowych typów obiektów, które pokrywają zapotrzebowanie na „często spotykane dane". Przykładowo, dostępne są oddzielne typy obiektów (modele danych) na dwuwymiarowe macierze do zapisu 8- i 24-bitowych obrazów monochromatycznych, wielokomponentowych obrazów kolorowych czy też 8-bitowych tablic kolorów (ang. *Lookup Table*). Dodatkowo, dostępne są również „uniwersalne" wielowymiarowe tablice oraz tabele. Takie specjalizowanie podstawowych typów danych ułatwia manipulację „standardowymi" danymi, jednocześnie ogranicza możliwość przechowywania danych o nietypowych parametrach i znacznie komplikuje interfejs programisty (API, ang. *Application Programming Interface*). Obiekty HDF4 mogą być koncepcyjnie przeniesione do modeli występujących w HDF5. Dodatkowo, nowa wersja zapewnia obiektowe podejście do zarządzania informacją o danych, ich atrybutach oraz typach. Ponieważ model obiektów w HDF5 został znacznie uproszczony, cały model danych jest bardzie spójny i prostszy. Implementacja operacji dostępu do danych wymaga jednak od programisty większej ilości kodu. Prawdopodobnie największym ograniczeniem w użyciu standardu HDF4 jest używanie w nim 32b adresowania, co prowadzi do ograniczenia maksymalnej wielkości pliku do 2GB. Wielkość taka może stanowić nieakceptowalne ograniczenie w przypadku dużych zbiorów danych.

Poza znacznymi różnicami w interfejsach programistycznych między starszą i nowszą wersją biblioteki, również pliki w wersji HDF4 i HDF5 nie są wzajemnie kompatybilne. Niezbędne jest konwertowanie plików, którą można wykonać przy użyciu programów *h4toh5* oraz *h5toh4*.

Ponieważ nowsza wersja standardu, oznaczana HDF5, jest częściej stosowana w nowych aplikacjach, dalsza część opisu standardu HDF dotyczy właśnie tej wersji.

### 2.6.1. Dokumentacja

Pełna dokumentacja standardu znajduje się na stronie organizacji *The HDF Group,* pod adresem `http://www.hdfgroup.org/documentation/`. Dokumentacja do wersji HDF5 podzielona jest na cztery główne grupy:

- dokumentację narzędzi: przeglądarki i edytora HDFView oraz licznej grupy programów narzędziowych uruchamianych z linii komend, umożliwiających: tworzenie, modyfikowanie, porównywanie oraz konwersję plików HDF;
- dokumentację ułatwiającą naukę:
  - samouczki (ang. *tutorials*)[9], które prezentują podstawy obsługi programu HDFView oraz liczne przykłady kodu źródłowego, tłumaczące podstawowe oraz zaawansowane aspekty wykorzystania biblioteki HDF;

[9] `http://www.hdfgroup.org/HDF5/Tutor/`

  - obszerną dokumentację dla programistów *Introduction to HDF5* [10];
  - słownik pojęć *HDF5 Glossary* [11];
- dokumentację biblioteki, która obejmuje przewodnik *HDF5 User's Guide* szczegółową dokumentację kodu biblioteki *HDF5 Reference Manual*, opis narzędzi interfejsu wyższego poziomu *High-level HDF5 Interfaces* oraz narzędzi dla C++ oraz Java;
- informacje dodatkowe, typu listę często zadawanych pytań i odpowiedzi (ang. *FAQs*), listę zmian wprowadzanych w kolejnych wersjach czy też zbiór dokumentów przydatny programistom aplikacji używającym biblioteki *HDF5 Application Developer's Guide* [12].

Na stronach internetowych *The HDF Group*, poza dokumentacją standardu, dostępne są również inne publikacje [13], takie jak: artykuły naukowe, prezentacje oraz dokumenty RFC (ang. *Requests for Commensts*). Dzięki temu, że hierarchiczny format danych HDF posiada dokumentację kierowaną do różnych użytkowników, od osób zainteresowanych jedynie odczytaniem danych po programistów aplikacji, format ten jest coraz częściej stosowany. W oparciu o HDF5 rozwijane są również inne biblioteki [14], specjalizowane do obsługi konkretnych typów danych np. *HDF-EOS* do danych z satelitów obserwujących Ziemię czy też *BioHDF* służące do przechowywania danych bioinformatycznych.

## 2.6.2. Organizacja plików – model danych

Pliki HDF, podobnie jak dokumenty XML, są „samoopisywalne" i pozwalają użytkownikowi na określenie złożonych relacji i zależności. W przeciwieństwie jednak do plików XML, pliki HDF mogą zawierać dane binarne, umożliwiają również bezpośredni dostęp do dowolnych danych, bez konieczności analizowania całej zawartości. Pliki HDF są tzw. pojemnikami (ang. *container*) na dowolne dane, które przechowywane są w sposób hierarchiczny.

Specyfikacja HDF5 posługuje się *modelami abstrakcyjnymi,* które umożliwiają zdefiniowanie oraz opisanie sposobu przechowywania i zarządzania skomplikowanymi danymi w plikach. Konkretną implementacją *abstrakcyjnego modelu danych* (ang. *Data Model*) oraz *abstrakcyjnego sposobu przechowywania danych* (ang. *storage model*), jest *biblioteka HDF5,* które stanowi gotowy interfejs programistyczny (API) obsługujący format HDF. Abstrakcyjne modele są niezależne od nośnika, systemu operacyjnego oraz środowiska programistycznego.

Dane w plikach HDF5 są zorganizowane w sposób hierarchiczny w postaci kierunkowego grafu. Węzły grafu są tworzone przy użyciu dwóch podstawowych struktur:

- grup (ang. *group*), które są strukturami zawierającymi zero lub więcej obiektów HDF, wraz z metadanymi (atrybutami, właściwościami) opisującymi zawartość,
- zbiorów danych (ang. *Data Sets*), będących wielowymiarowymi tablicami elementów danych, wraz z metadanymi.

[10] http://www.hdfgroup.org/HDF5/doc/H5.intro.html
[11] http://www.hdfgroup.org/HDF5/doc/Glossary.html
[12] http://www.hdfgroup.org/HDF5/doc/ADGuide.html
[13] http://www.hdfgroup.org/pubs
[14] http://www.hdfgroup.org/projects/

Obiektami HDF mogą być inne grupy lub zbiory danych. Każda grupa HDF5 lub zbiór danych HDF5 mogą posiadać przypisane listy *atrybutów,* które zapewniają dodatkowe informacje o obiekcie i są definiowane przez użytkownika. Każdy plik HDF5 zawiera przynajmniej jeden obiekt, którym jest *grupa początkowa* (ang. *Root Group*), która tworzy korzeń grafu który oznaczany jest znakiem „/", tak jak początek systemu plików w Unix. Wszystkie inne obiekty przechowywane w pliku są potomkami grupy początkowej. Obiekty posiadają przypisane unikatowe identyfikatory *nazwy,* dzięki którym możliwy jest do nich dostęp. Do konkretnych obiektów dostęp możliwy jest jedynie poprzez ścieżkę dostępu, która odzwierciedla hierarchię obiektów.

**Grupa danych**

Grupa danych HDF5 jest strukturą zawierającą zero lub więcej obiektów, składa się z dwóch części:

- nagłówka grupy, który zawiera nazwę grupy oraz listę atrybutów grupy,
- tabeli symboli obiektów, która zawiera listę obiektów należących do grupy.

Praca z grupami i elementami grup jest podobna do pracy z katalogami i plikami w systemie Unix – obiekty HDF5 posiadają nazwy oraz „ścieżkę dostępu", która opisuje „adres" obiektu, hierarchiczne zależności między obiektami. Tak jak w „prawdziwym" systemie plików, również w hierarchicznej strukturze obiektów HDF mogą występować *dowiązania symboliczne* między obiektami oraz „podmontowane" obiekty, które fizycznie znajdują się w innych plikach.

**Zbiór danych**

Zbiór danych HDF5 jest tablicą *elementów danych,* ułożonych według przyjętej specyfikacji (ang. *dataspace*). Element danych jest najmniejszą adresowalną jednostką danych przechowywaną w pliku HDF5. Zbiory danych przechowywane są w plikach, i można wyróżnić w nich na dwie części: nagłówek oraz tablicę danych.

W nagłówku można rozróżnić cztery podstawowe rodzaje informacji:

1. Nazwę (ang. *name*) zbioru danych, którą tworzy ciąg znaków ASCII.
2. Typy danych (ang. *datatype*), które dzielą się na dwie kategorie: typy podstawowe (ang. *atomic*) oraz złożone. Typy podstawowe, takie jak liczby całkowite lub zmiennoprzecinkowe, posiadają własne parametry określające: długość, precyzję czy też uwzględnienie znaku. Z typami podstawowymi związane są typy wbudowane (ang. *native*), zależne od konkretnego systemu komputerowego, będące specyficznymi przypadkami typów atomowych. Przykładem typów wbudowanych HDF5 są: H5T_NATIVE_CHAR, H5T_NATIVE_INT czy H5T_NATIVE_DOUBLE. Pełną listę wbudowanych typów można znaleźć w dokumentacji biblioteki [15].
   Typy złożone składają się z typów podstawowych, posiadają budowę zbliżoną do *struktury* (ang. *struct*) znanej z języka C. Specyficznym dla HDF5 typem danych są *nazwane typ danych* (ang. *named datatypes*), które umożliwiają rozdzielenie definicji typu danych od samych danych.

[15] `http://www.hdfgroup.org/HDF5/doc/doc-info.html`

Definicja *nazwanego* typu danych przechowywana jest w oddzielnym pliku, niezależnie od samych danych. Umożliwia to ich wielokrotne wykorzystywanie dla wielu plików z danymi.

3. Wymiar danych (ang. *dataspace*), może być określony bez możliwości zmiany (ang. *fixed*) lub też może być „nieograniczony", co oznacza, że może być w dowolnym momencie zmieniony. W tej części nagłówka przechowywane są informacje o liczbie wymiarów macierzy danych, aktualnych oraz maksymalnych wymiarach tablicy. Wymiar danych może być również określony przez tzw. *zaznaczenie* (ang. *selections*), które umożliwia dostęp do części danych.
   Zaznaczenie umożliwia wybranie z *n*-wymiarowych danych, n-wymiarowego fragmentu (bloku) logicznie połączonych danych. Umożliwia również wybranie bloku danych nie sąsiadujących ze sobą lecz równo rozmieszczonych. Dostępne są mechanizmy, umożliwiające łączenie wybranych bloków danych oraz wybieranie listy pojedynczych punktów. Mechanizm ten jest szczególnie przydatny w przypadku dużych zbiorów danych wielowymiarowych. Szczegółowy opis można znaleźć w dokumentacji.
4. Sposób przechowywania danych (ang. *storage layout*) określa organizację danych. Domyślnym formatem przechowywania danych jest *układ ciągły* – dane są przechowywane w ten sam sposób jak w pamięci operacyjnej. Dostępne są również dwa dodatkowe tryby: *kompaktowy* (ang. *compact*), szczególnie użyteczny w przypadku małej ilości danych w tablicach (tzw. tablice rzadkie) oraz tryb danych pofragmentowanych (ang. *chunked*) w którym zbiór danych dzielony jest na przechowywane oddzielnie fragmenty o jednakowej wielkości. Dzięki temu możliwa jest np. skuteczne zwiększanie wymiarów zestawu danych w dowolnym kierunku lub też kompresja dużych zbiorów danych przy jednoczesnym zachowaniu wysokiej wydajności podczas dostępu.

**Atrybuty danych**

Atrybuty (ang. *attribute*) są dołączane do podstawowych zestawów grup lub nazwanych typów danych (ang. *named datatype*). Używane są w celu opisania charakteru oraz przeznaczenia zestawu danych lub grupy. Stanowią więc dokumentację obiektu HDF5. Składają się z dwóch części: *nazwy* (ang. *name*) oraz *wartości* (ang. *value*). Część zawierająca wartości może zawierać jeden lub więcej wpisów danych tego samego typu.

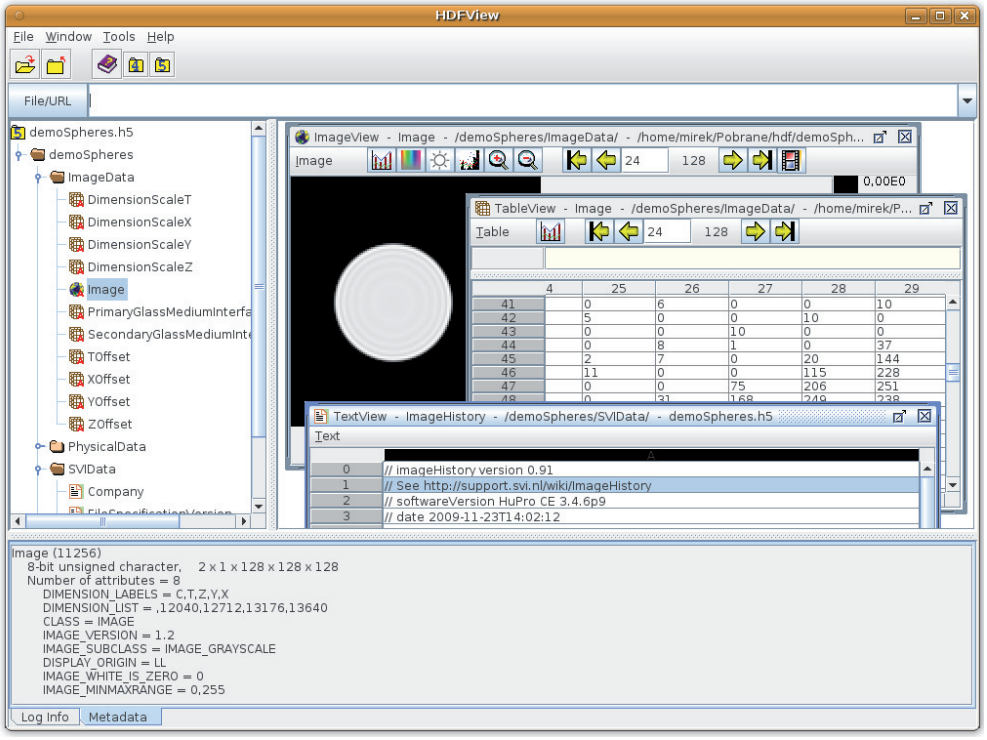
## 2.6.3. Przeglądarka danych HDF – HDFView

Program HDFView można pobrać ze strony *The HDF Group*[16] jest on również dostępny w repozytoriach systemów Linux. Program został napisany w języku Java i jest rozprowadzany wraz z niezbędnymi do uruchomienia bibliotekami. Interfejs programu przedstawiono na rysunku 2.8.

HDFView umożliwia tworzenie, przeglądanie oraz edycję plików w formacie HDF4 oraz HDF5. Hierarchia danych przedstawiana jest w postaci drzewa obiektów. Typ obiektu symbolizowany jest odpowiednią ikoną, zaś szczegółowe informacje o metadanych znajdują się w dolnej zakładce programu. Program ma wbudowane narzędzia do przeglądania

[16] `http://www.hdfgroup.org/hdf-java-html/hdfview/`

zawartości obiektów. Na rysunku 2.8 przedstawiono podgląd danych `/demoSpheres/ImageData/Image` w tabeli liczb (TableView) oraz w postaci graficznej (ImageView). Dostępna jest również wizualizacja danych w postaci dwuwymiarowego wykresu wielu serii danych. Program umożliwia tworzenie nowych plików, dodawani oraz kasowanie grup i zbiorów danych, modyfikowanie atrybutów oraz zawartości zbiorów danych.




**Rys. 2.8.** Interfejs programu HDFView z wczytanymi przykładowymi danymi

Program ma rozbudowaną dokumentację, która w przystępny sposób opisuje jego możliwości. Przykład zastosowana programu HDFView do tworzenia pliku HDF5 bez potrzeby programowania, znajduje się na stronie `http://www.hdfgroup.org/HDF5/Tutor/hdfview.html`.

## 2.6.4. Interfejs programistyczny – HDF5 API

Biblioteka HDF5 została napisana w języku C i może być dołączana do programów napisanych w różnych językach. Umożliwia tworzenie plików w formacie HDF5, w tym tworzenie i zapis grup, zbiorów danych oraz atrybutów. Wykorzystywana jest więc w aplikacjach naukowych jako warstwa oprogramowania odpowiedzialna za składowanie i dostęp

do hierarchicznie powiązanych danych. Tworzenie, zarządzanie oraz interpretacja hierarchii danych jest zadaniem „wyższych" warstw aplikacji. Sama biblioteka HDF5 wywołuje dostarczane przez system operacyjny lub inne biblioteki, metody odpowiedzialne za operacje wejścia-wyjścia dotyczące plików. HDF5 wykorzystuje tzw. sterowniki plików (ang. *file driver*), które implementują różne metody dostępu do plików[17]. Dostępne są między innymi klasyczny interfejs buforowanego dostępu do pojedynczego pliku (oparty na stdio.h), bardzo szybkie przechowywanie plików w pamięci operacyjnej oraz interfejs umożliwiający równoległy dostęp wielu aplikacji do jednego pliku, oparty na protokole MPI (ang. *Message Passing Interface*). Biblioteka HDF5 współpracuje również z tzw. bibliotekami filtrów, które odpowiadają np. za kompresję zapisywanych danych.

**Interfejs niskiego poziomu**

Wszystkie nazwy procedur w bibliotece HDF5 zaczynają się od symbolu H5, po którym mogą wystąpić litery oznaczające rodzaj obiektu, na którym dana funkcja działa. Takie nazewnictwo umożliwia „symulowanie obiektowości" biblioteki, niedostępnej w języku C.

W bibliotece HDF5 można wyróżnić następujące grupy funkcji, które tworzą tzw. *interfejsy niskiego poziomu* (ang. *low-level APIs*)*:*

H5: procedury podstawowe, ogólnego zastosowania, np. takie jak otwieranie i zamykanie plików (H5open, H5close) czy też weryfikację wersji (H5get_libversion).

H5A: funkcje dotyczące atrybutów (ang. *attribute*), umożliwiają tworzenie (H5Acreate) dostęp (np. H5Aopen, H5Aclose, H5Aread, H5Awrite) i manipulację atrybutami danych (np. H5Adelete, H5Arename).

H5D: funkcje dotyczące zbiorów danych (ang. *dataset*), takie jak tworzenie oraz dostęp do wartości (np. H5Dcreate, H5Dget_type).

H5E: funkcje umożliwiające obsługę błędów (ang. *error*), umożliwiające przykładowo wydruk błędów (np. H5Eprint).

H5F: funkcje dostępu do plików (ang. *file*), obejmujące operacje podstawowe, takie jak otwieranie, zamykanie plików (np. H5Fopen, H5Fclose, H5Fmount), jak również dostęp do informacji nagłówkowych pliku (np. H5Fget_name, H5Fget_info).

H5G: funkcje umożliwiające zarządzanie grupami danych (ang. *group*), możliwe jest tworzenie, kopiowanie, otwieranie, zamykanie oraz zarządzanie listą właściwości.

H5I: funkcje zarządzające identyfikatorami obiektów (ang. *identifiers*).

H5L: funkcje służące do tworzenia i zarządzania połączeniami między obiektami danych (ang. *links*).

H5O: funkcje do zarządzania obiektami danych (ang. *object*).

H5P: funkcje do obsługi list właściwości (ang. *property list*).

H5R: funkcje umożliwiające tworzenie połączeń symbolicznych (ang. *reference*) w obrębie pliku HDF5.

[17] Szczegółowe informacje można znaleźć w rozdziale Virtual File Layer API dokumentacji.

H5S: funkcje zarządzające organizacją danych (ang. *dataspace*).

H5T: funkcje służące do tworzenia oraz zarządzania typami danych (ang. *datatypes*).

H5Z: funkcje obsługujące filtry, umożliwiające np. kompresję danych lub też obliczanie i sprawdzanie sum kontrolnych.

**Interfejs wysokiego poziomu**

W celu ułatwienia korzystania z biblioteki HDF5, powstały również procedury tworzące tzw. *interfejs wyższego poziomu* (ang. *high-level APIs*). Zestaw tych funkcji korzysta z funkcji niższego poziomu. Interfejs wyższego poziomu powstał w celu ujednolicenia podejścia do najczęściej wykorzystywanych przypadków użycia biblioteki HDF5. Można wyróżnić następujące grupy funkcji, które ułatwiają korzystanie z biblioteki HDF5:

H5LT: *HDF5 Lite API,* podstawowe funkcje interfejsu wyższego poziomu, umożliwiające „jednolinijkowe" zarządzanie:
- zbiorami danych, obejmujące tworzenie (H5LTmake_dataset), czytanie (H5LTread_dataset) oraz przeszukiwanie (H5LTfind_dataset),
- typami danych (H5LTtext_to_dtype i H5LTdtype_to_text), które umożliwiają konwersję nazwy typów danych na odpowiadające im identyfikatory,
- atrybutami, takie jak wpisywanie wartości atrybutu (H5LTset_attribute_string), odczytywanie (H5LTget_attribute) oraz przeszukiwanie (np. H5LTfind_ attribute),

H5IM: *HDF5 Image API,* zestaw funkcji do obsługi danych typu bitmapa, obejmujący między innymi tworzenie obrazów (np. H5IMmake_image_8bit) i palet kolorów (H5IMmake_palette).

H5TB: *HDF5 Table API,* funkcje do obsługi danych typu tabela: tworzenie, modyfikowanie, wyszukiwanie, zapis i odczyt.

H5PT: *HDF5 Packet Table API,* funkcje do wydajnej obsługi danych typu tabela, bez możliwości swobodnego dodawania i kasowania danych.

H5DS: *HDF5 Dimension Scale API,* funkcje ułatwiające zarządzanie dodatkowymi informacjami przypisywanymi do informacji o wymiarach, takim jak np. opisy poszczególnych wymiarów danych.

H5LR: różne rozszerzenia funkcjonalności, które obejmują narzędzia do zarządzania odnośnikami (ang. *region references*), obsługę dostępu do podzbiorów wyznaczanych przez wielowymiarowe „zaznaczenia" (ang. *hyper slab selections*) oraz narzędzia umożliwiające dostęp do danych na poziomie bitów (ang. *bit-fields*).

Wykorzystanie funkcji interfejsu wysokiego poziomu znacznie ułatwia implementację hierarchicznego standardu danych HDF5 we własnych aplikacjach. Praktyczne przykłady zastosowania funkcji API wysokiego poziomu w programach, można znaleźć na stronie `http://www.hdfgroup.org/HDF5/Tutor/highlevel.html` oraz na innych stronach dokumentacji.

Pełna dokumentacja HDF5 API znajduje się na stronie organizacji *The HDF Group* pod adresem: `http://www.hdfgroup.org/HDF5/doc/EM/EM_H5Front.html`.

### 2.6.5. HDF w programie Matlab

Program Matlab ma liczny zestaw funkcji, które umożliwiają pełen dostęp do danych zapisanych w standardzie HDF4 oraz HDF5. Tak jak to miało miejsce w przypadku biblioteki HDF, również w Matlabie podzielono funkcje obsługujące standard na dwie grupy: interfejs wysokiego oraz niskiego poziomu.

Do funkcji wysokiego poziomu można zaliczyć trzy podstawowe funkcje:

1. `hdfinfo`, `hdf5info`: dwie wersje funkcji, dla plików HDF4 oraz HDF5, które umożliwiają odczytanie informacji o zawartości wskazanego pliku,
2. `hdfread`, `hdf5read`: funkcje umożliwiające odczytanie całej zawartości pliku oraz danych należących do wskazanego nazwą zbioru danych lub atrybutu,
3. `hdfwrite`, `hdf5write`: funkcje umożliwiają zapis wskazanych danych do konkretnego pliku z uwzględnieniem hierarchicznych zależności między danymi.

W interfejsie niskiego poziomu, Matlab oferuje bezpośredni dostęp do kilkuset funkcji zawartych w bibliotece HDF5 oraz kilkanaście specjalizowanych skryptów do obsługi standardu HDF4. Dzięki możliwości bezpośredniego wywołania funkcji biblioteki HDF5, możliwe jest tworzenie bardzo zaawansowanych skryptów, które mogą odczytywać lub zapisywać dane w formacie HDF5 z wykorzystaniem wszystkich możliwości biblioteki. Poszczególne grupy funkcji interfejsu niskiego poziomu biblioteki HDF5 zostały zgrupowane w taki sposób, że trzyliterowy przedrostek nazwy funkcji służy do identyfikacji pakietów funkcji Matlaba. Przykładowo, wywołanie funkcji `H5F.close` Matlaba, odpowiada funkcji `H5Fclose` biblioteki HDF5. Składnia oraz lista parametrów funkcji w większości się nie zmieniła, jedynie w kilku przypadkach uległa drobnym zmianom. W ten sposób w programie Matlab dostępne są wszystkie funkcje biblioteki, zgrupowane w następujących pakietach: H5, H5A, H5D, H5DS, H5E, H5F, H5G, H5I, H5L, H5ML, II5O, H5P, H5R, H5S, H5T, H5Z oraz kilka dodatkowych funkcji, zawartych w H5ML, specyficznych dla środowiska Matlab. Dokumentacja poszczególnych pakietów i funkcji dostępna jest poprzez wywołanie funkcji pomocy: `help H5F` lub help `H5F.open`.

Więcej informacji na temat współpracy programu Matlab z naukowym formatem danych HDF, można znaleźć w dokumentacji programu (polecenie: `doc hdf5`), na stronach producenta programu Matlab z dokumentacją on-line `http://www.mathworks.com/help/techdoc/ref/hdf5.html`, oraz stronach *The HDF Group* z przykładowymi skryptami `http://www.hdfgroup.org/ftp/HDF5/examples/examples-by-api/api18-m.html`.

### 2.6.6. Podsumowanie

Hierarchiczny format zapisu danych naukowych HDF, szczególnie w najnowszej wersji HDF5, może stanowić bardzo dobrą bazę to tworzenia własnych aplikacji, które wymagają wydajnego zarządzania dużymi zbiorami danych, wymagających uwzględnienia skomplikowanych wzajemnych relacjach oraz o rozbudowanych typach danych. Biblioteka HDF5 stanowiąca implementację standardu, oferuje rozbudowane narzędzia programistyczne, zarówno wysokiego, jak i niskiego poziomu, które ułatwiają implementację składowania danych. Niewątpliwą zaletą standardu jest jego otwartość oraz elastyczność, które umożliwiają

wykorzystanie jej możliwości w praktycznie dowolnym zadaniu. Również szerokie wsparcie w programach komercyjnych typu Matlab może być zachętą do przechowywania danych nawet mniej skomplikowanych w formacie HDF5.

Z drugiej strony, pomimo ogromnej elastyczności i możliwości dostosowania typów danych, zastosowanie biblioteki HDF5 we własnym kodzie wymaga zazwyczaj zaprojektowania i zdefiniowanie własnych typów danych, które będą użyte do przechowywania danych. Dodatkowo, należy zaprojektować i zaimplementować tworzenie i zarządzanie hierarchią obiektów przechowujących dane. W przypadku „skomplikowanych danych" wynikających np. z zastosowanych innych narzędzi informatycznych, stworzenie pomostu między danymi „w pamięci komputera" oraz strukturami HDF może wymagać sporego nakładu pracy programistów.

# Rozdział 3
# Biblioteka Visualization Toolkit

Biblioteka *Visualization Toolkit*, w skrócie VTK, jest bardzo rozbudowanym narzędziem do przetwarzania oraz wizualizacji danych. Jest to biblioteka wieloplatformowa, rozpowszechniana w postaci kodu na zasadach wolnego oprogramowania (ang. *open-source*). Napisano ją w języku C++ z bardzo intensywnym wykorzystaniem możliwości programowania zorientowanego obiektowo. Biblioteka zawiera interfejsy do innych języków, takich jak Tcl/Tk, Java oraz Python i jest niezależna od graficznego interfejsu użytkownika (GUI). Zaimplementowano w niej wsparcie do popularnych bibliotek, takich jak: MFC, Qt czy Tk. Dzięki temu, możliwe jest umieszczenie kodu VTK w praktycznie dowolnym programie.

Biblioteka VTK nie jest programem, nie da się jej „uruchomić", można ją co najwyżej użyć. Jest zbiorem kilkuset obiektów, które po odpowiednim połączeniu tworzą program. Można jej użyć zarówno w programie typu konsola, jak i w programach okienkowych. Biblioteka ta do wyświetlania grafiki używa biblioteki OpenGL wykorzystującej sprzętową akceleracje tworzenia grafiki w kartach graficznych. Dzięki temu zapewnia bardzo dobrą wydajność oraz przenośność kodu.

Biblioteka VTK powstała w firmie *Kitware*[1] i jest obecnie rozwijana przez szerokie grono ochotników oraz specjalistów z dziedziny grafiki komputerowej, wizualizacji, przetwarzania równoległego i wielu innych. Zespół pracowników firmy Kitware nadal nadzoruje rozwój biblioteki VTK oraz innych projektów[2], w tym: *VTKEdge* będącego rozszerzeniem funkcjonalności VTK o zaawansowane przetwarzania na procesorach kart graficznych (ang. *graphic-procesor-unit computing*) oraz *ITK* (ang. *Insight Segmentation and Registration Toolkit*), rozbudowanej biblioteki do przetwarzania głównie danych medycznych. Na bazie powyższych bibliotek powstały dwa programy: *ParaView* udostępniony na zasadach wolnego oprogramowania oraz komercyjny *VolView*.

Kod źródłowy najnowszej wersji VTK dostępny jest pod adresem `http://www.vtk.org/VTK/resources/software.html#latest`. Oprócz spakowanego kodu źródłowego, udostępnione są również, w postaci archiwum, przykładowe dane oraz dokumentacja. Użytkownicy systemu Windows mają do dyspozycji skompilowaną wersję w postaci instalatora. Po instalacji możliwe jest uruchomienie interpretatora języka Tcl, w którym można przetestować działanie biblioteki. Dla użytkowników systemu Linux, w szczególności dystry-

[1] Strona domowa firmy: `http://www.kitware.com`

[2] Lista niekomercyjnych projektów: `http://www.kitware.com/opensource/opensource.html`

bucji Ubuntu, przygotowane zostały tzw. *pakiety,* czyli kompatybilne z dystrybucją wersje biblioteki. W celu instalacji VTK w systemie Ubuntu, należy uruchomić program *Menadżer pakietów Synaptic.* Następnie należy wyszukać w repozytoriach pakiety zawierające frazę „vtk" i zainstalować: `libvtk5`, `vtkdata`, `vtk-doc`, `vtk-examples`, `vtk-tcl`, `python-vtk`, `libvtk5-dev`, `libvtk5-qt4`, `libvtk5-qt4-dev` wraz ze wszystkimi zależnymi pakietami, które zaproponuje Synaptic.

Dostęp do kodu źródłowego biblioteki, umożliwia skompilowanie jej w (prawie) dowolnym systemie operacyjnym oraz z innymi niż standardowe parametrami. Do przeprowadzania własnej kompilacji potrzebny jest program *cmake,* dostępny w wersji dla wielu platform na stronie `http://www.cmake.org`. Program można również znaleźć w repozytoriach dla Ubuntu (pakiety: `cmake` oraz `cmake-qt-gui`).

## 3.1. Dokumentacja

Podstawowym źródłem wiedzy na temat biblioteki VTK jest bardzo obszerna dokumentacja *on-line,* dostępna pod adresem `http://www.vtk.org/VTK/help/help.html`. Pod tym adresem znajdują się odnośniki do:

- dokumentacji kodu źródłowego, wygenerowanej przez program Doxygen na podstawie kodu i zawartych w nim komentarzy, jest to podstawowe źródło wiedzy podczas programowania; zawiera opis klas (alfabetyczny, listę oraz hierarchię), a dla każdej klasy: krótki opis, graf dziedziczenia, graf współpracy klas, listę metod wraz z ich opisem oraz odnośniki do podobnych klas i przykładowych programów (głownie testujących),
- listy najczęściej zadawanych pytań: `http://www.cmake.org/Wiki/VTK/FAQ`,
- dokumentacji typu wiki: `http://www.vtk.org/Wiki/VTK`,
- przykładowych programów: `http://www.vtk.org/Wiki/VTK/Examples`,
- listy dyskusyjnej dla użytkowników i programistów.

Zostały również wydane dwie książki *The Vizualization Toolkit* [SML04] oraz *VTK User's Guide* [AbK104]. Opisują one teoretyczne podstawy zaimplementowanych algorytmów oraz architekturę biblioteki VTK. Kierowane są do użytkowników oraz programistów pragnących rozszerzać funkcjonalność VTK.

## 3.2. Architektura

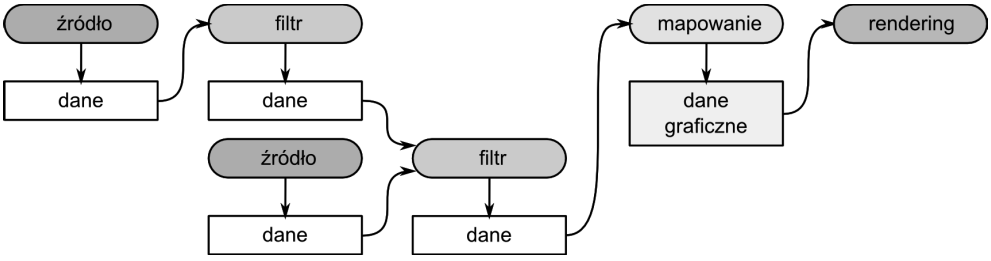
VTK jest biblioteką zorientowaną obiektowo, wykorzystującą paradygmat programowania obiektowego, w którym to program składa się z abstrakcyjnych *obiektów* łączących w sobie *dane* (atrybuty) z *metodami* (funkcjami), odpowiedzialnymi za przetwarzanie danych oraz komunikację z innymi obiektami. Takie podejście do programowania jest zgodne z intuicyjnym postrzeganiem rzeczywistych obiektów, które postrzegamy jako całość. Rzeczywiste obiekty posiadają atrybuty (cechy), które je definiują (np. ołówek ma rozmiary, kolor, twardość grafitu itp.) oraz funkcje, działania, które można wykonać *przy pomocy* lub

*na* obiekcie (np. ołówek można zastrugać, co spowoduje zmianę jego rozmiarów, a następnie użyć wraz z kartką, czyli innym „obiektem", do zapisania notatki).

Biblioteka VTK składa się z kilku tysięcy klas[3] języka C++, intensywnie wykorzystujących możliwości tego języka. Niskopoziomowe części biblioteki, takie jak: struktury danych, algorytmy oraz funkcje wymagające dużej wydajności, zaimplementowane są w „czystym C++". Powszechnie wykorzystywane są również *wzorce projektowe* (ang. *design pattern*), takie jak *fabryki obiektów* (ang. *object factories*), *obserwatory* (ang. *observer*), *iteratory* (ang. *iterator*) oraz funkcje wirtualne (ang. *virtual functions*) zapewniające polimorfizm obiektów. W wyniku tego biblioteka jest wydajna, łatwa w rozbudowie oraz niezależna od sprzętu.

### 3.2.1. Strumień wizualizacji

Wizualizacja danych zaimplementowana w VTK wykorzystuje podejście strumieniowe, w którym dane przepływają między połączonymi obiektami. Przepływ danych (ang. *data flow*) zaczyna się od wejścia, jakim są obiekty *źródła* (ang. *source*) wczytujące lub generujące dane, poprzez obiekty *filtry* (ang. *filter*) odpowiedzialne za przetwarzanie danych, do obiektów *mapujących* (ang. *mapper*) wyniki przetwarzania do postaci podstawowych obiektów geometrycznych (np. punktów, linii) oraz ich parametrów (np. położenia, koloru czy rozmiaru). Na samym końcu strumienia znajdują się obiekty odpowiedzialne za generowanie obrazu (ang. *rendering*), czyli transformowania danych geometrycznych do postaci graficznej. Rysunek 3.1 przedstawia przykładowy strumień wizualizacji (ang. *visualization pipeline*), zawierający: dwa źródła, filtry o jednym oraz dwóch wejściach, obiekt mapujący oraz renderujący obraz.




**Rys. 3.1.** Wzajemnie połączone obiekty tworzą strumień wizualizacji danych

Obiekty tworzące strumień wizualizacji danych można podzielić na dwie kategorie:

1. Wywodzące się z klasy `vtkAlgorithm` obiekty przetwarzające jedne dane w inne:
   – źródła strumienia danych `vtkSource`, które obejmują obiekty wczytujące dane z plików i innych kanałów przesyłu informacji (np. ze strumienia sieci) oraz obiekty generujące dane (np. współrzędne wierzchołków sfery o zadanej średnicy i liczbie wierzchołków),

[3] `http://www.vtk.org/doc/release/5.6/html/classes.html`

- filtry `vtkProcessObject`,
- obiekty tworzące dane graficzne `vtkMapper`, zarówno dwu-, jak i trójwymiarowe, stanowią pomost między strumieniem wizualizacji a częścią biblioteki odpowiedzialną za graficzną prezentację wyników ich działania,
- inne ujścia strumienia danych `vtkWriter`, głównie obiektów zapisujących dane do plików, które można potraktować jako specyficzne obiekty mapujące dane.

2. Będące potomkami obiektu `vtkDataObject` obiekty przechowujące abstrakcyjne dane, z uwzględnieniem wzajemnych powiązań (topologii) oraz współrzędnych geometrycznych.

Obiekty łączy się ze sobą za pomocą metod `SetInput()` oraz `GetOutput()`, np. wywołanie `A.SetInput(B.GetOutput())` spowoduje połączenie wyjścia obiektu B do wejścia obiektu A. Źródła danych nie posiadają wejść (ang. *input*), za to mogą udostępniać co najmniej jedno wyjście (ang. *output*). Z kolei filtry mogą mieć wiele wejść oraz wyjść. Dodatkowo występują dwa typy wielokrotnych wyjść filtrów: dostarczające takie same dane do wielu obiektów (ang. *fan out*) oraz dostarczające różne dane da różnych wyjść (ang. *multiple output*). Obiekty mapujące oraz ujścia strumienia wizualizacji mają przynajmniej jedno wejście i nie posiadają wyjść.

Dane, które „płyną" w strumieniu wizualizacji, mogą być różnego typu, dlatego konieczne jest sprawdzanie, czy łączone ze sobą wejścia oraz wyjścia obiektów są ze sobą kompatybilne. Zgłaszanie błędów odbywa się podczas kompilacji programu w języku C++ lub podczas wykonywania programu w przypadku Tcl. VTK udostępnia również zaawansowane mechanizmy umożliwiające zarządzanie przechowywaniem danych w pamięci komputera. Jest to szczególnie ważne podczas wizualizacji dużych zbiorów danych, których rozmiary często przekraczają wielkość dostępnej pamięci operacyjnej. Możliwe jest włączenie lub wyłączenie wykorzystania pamięci podręcznej (ang. *cache*) oraz ograniczenie liczby kopii danych poprzez mechanizm zliczania referencji (ang. *reference counting*). Umożliwia on wykorzystanie jednej kopii danych jednocześnie przez kilka obiektów. Zaimplementowano również dynamiczne zarządzanie pamięcią, które umożliwia natychmiastowe zwalnianie nieużywanych zasobów, oczywiście jeśli jest to tylko możliwe. Przykładowo, filtry zaraz po wykonaniu obliczeń na danych wejściowych oraz udostępnieniu wyników „na wyjście", zezwalają na zwolnienie (skasowanie) pamięci danych wejściowych. W takim podejściu, w pamięci przechowywane są jedynie bieżące dane. Wiąże się to jednak ze zwiększonymi kosztami obliczeniowymi – w strumieniu wizualizacji nie są dostępne „dane pośrednie". Trzeba je za każdym razem wyliczać na nowo. W bibliotece VTK dostępne jest również statyczne zarządzanie pamięcią, które ogranicza potrzebę wielokrotnego dokonywania obliczeń pośrednich, jednak sprawdza się ono tylko przy przechowywaniu relatywnie małych porcji danych. Ostatnim mechanizmem wpływającym na wydajność i ograniczającym zapotrzebowanie na pamięć jest mechanizm strumieniowania danych, umożliwiający dzielenie danych na mniejsze części, podlegające obliczeniom. Mechanizm ten jest bardzo wydajny i umożliwia wykonywanie np. równoległych obliczeń na wielu niezależnych komputerach.

VTK przetwarza dane tylko wtedy, gdy są one potrzebne. Oznacza to, że samo połączenie w strumień obiektów nie „uruchomi" strumienia. Zarządzanie uruchamianiem poszczególnych obiektów strumienia odbywa się na podstawie przypisywanego indywidualnie dla

poszczególnych obiektów tzw. *czasu modyfikacji obiektu.* Dzięki temu, możliwe jest określenie, czy konkretny obiekt ma dane nadal aktualne w porównaniu z innym obiektem. Przykładowo, jeżeli stworzymy w programie obiekt wczytujący dane z pliku, ustalimy poprawną ścieżkę dostępu do pliku, a następnie wywołamy metodę zwracającą np. rozmiary wczytywanych danych, w wyniku otrzymamy wartość 0. Dzieje się tak dlatego, że obiekt taki nie został uruchomiony – owszem już istnieje w pamięci komputera, jest gotowy do działania, jednak jeszcze nie wczytał danych. Wszystkie obiekty tworzące strumień wizualizacji posiadają metodę `Update()`, której zadaniem jest właśnie „uruchamianie" działania obiektów. Jeśli więc przed wywołaniem metody zwracającej wymiary danych zostanie wywołana metoda `Update()`, wówczas obiekt wczyta dane, a następnie zwróci rzeczywiste rozmiary danych zawartych w pliku.

Zazwyczaj nie ma potrzeby ręcznego uruchamiania obiektów, czyli wywoływania ich implementacji `Update()`, ponieważ odpowiadają za to automatyczne mechanizmy strumienia wizualizacji. Jak już wcześniej wspomniano, VTK przetwarza dane (uruchamia obiekty) tylko wtedy, gdy te dane są potrzebne. „Zapotrzebowanie" automatycznie zgłaszają obiekty, znajdujące się na końcu strumienia: mapujące lub zapisujące dane. Przepływ danych odbywa się w kierunku: źródła → filtry → obiekty mapujące → obiekty generujące obraz, zaś uruchamianie przetwarzania danych odbywa się w kierunku przeciwnym. Obiekty renderujące wywołują metodę `Update()` obiektów mapujących, te z kolei zgłoszą zapotrzebowanie do podłączonych do nich filtrów, a te do źródeł danych. W ten sposób cały strumień danych zostaje uruchomiony.

Jeżeli obiekt (np. filtr) znajdujący się „w środku" strumienia wizualizacji zmieni dane np. na skutek zmian w nastawach parametrów realizowanego algorytmu, wówczas obiekt taki wywołuje metodę `ExecuteData()` obiektów podłączonych do swoich wyjść. Metoda ta powiadamia obiekty o aktualizacji danych w „przeciwnym kierunku" niż metoda `Update()`.

Podsumowując, biblioteka VTK ma dwa tryby uruchamiania strumienia wizualizacji:

- *Zależny od zapotrzebowania na dane* (ang. *demand-driven*) – realizowany „od końca" strumienia za pomocą metody `Update`, tylko gdy wyjście danego obiektu jest potrzebne. Dzięki temu, uruchamiana jest wyłącznie niezbędna część strumienia. Umożliwia to wykonywanie wielu zmian w strumieniu, bez konieczności wielokrotnego przeliczania danych. Ogranicza to zbędne obliczenia, co jest bardzo istotne w programach dostarczających interaktywnej wizualizacji danych.
- *Sterowany zdarzeniami* (ang. *event-driven*) – realizowany do końca strumienia metodą `ExecuteData()`, której główną zaletą jest zapewnianie aktualności danych w strumieniu.

Przy okazji przejścia od wersji VTK 4.0 do VTK 5.0, programiści przebudowali mechanizmy tworzenia, zarządzania i wykonywania strumienia wizualizacji. Nowy mechanizm jest kompatybilny wstecznie, działa jednak sprawniej i zapewnia jeszcze większą elastyczność[4]. Podstawą nowego podejścia jest łączenie filtrów przy pomocy metod: `B->SetInputConnection(A->GetOutputPort())`. Różnica jest taka, że metoda `OutputPort()` (pochodząca od klasy `vtkAlgorithmOutput`) umożliwia realizację połączenia

[4] Szczegółowe informacje: `http://www.cmake.org/Wiki/VTK/FAQ`

typu: „*N*-ty port *X*-owego algorytmu" obiektu A. Zazwyczaj $N = 0$, jednak można w jednym obiekcie w prosty sposób udostępnić wiele *portów,* przez które można dostarczyć różne dane z różnych części algorytmu. Zaś metoda `Output()` zwraca tylko jeden kontener (ang. *container*) na dane. Tylko obiekty potomne po `vtkAlgorithm` mogą posiadać metodę `SetInputConnection()` lub `GetOutputPort()`. Istnieje spora rodzina obiektów, zwłaszcza w części VTK odpowiedzialnej za rendering obrazu, które nie posiadają „nowych" metod. Dla takich obiektów należy używać metod „starych", czyli `SetInput()` i `GetOutput()`. W pozostałych przypadkach zalecane jest używanie nowej architektury, w wersjach nowszych od VTK 5.0.

## 3.2.2. Model graficzny

Jak już wiadomo, strumień wizualizacji (ang. *visualization pipeline*) stanowi podstawę biblioteki VTK. Drugim bardzo ważnym podsystemem jest część odpowiedzialna za *tworzenie i prezentację danych graficznych* w postaci cyfrowego obrazu. Ta część biblioteki, nazywana *modelem graficznym,* (ang. *graphics model*), stanowi abstrakcyjną warstwę pomiędzy strumieniem wizualizacji, a niskopoziomowymi interfejsami lub językami graficznymi.

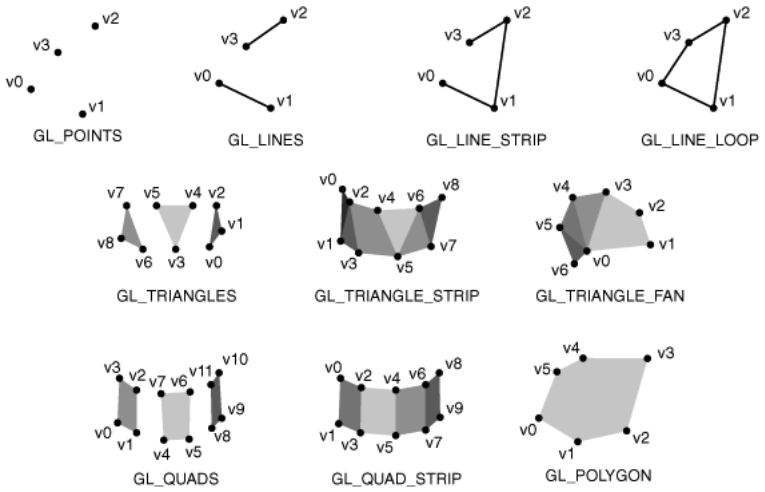
Podstawowym interfejsem graficznym dla VTK jest obecnie OpenGL. Jest to niezależna od systemu operacyjnego oraz sprzętu biblioteka, która do generowania cyfrowych obrazów wykorzystuje możliwości sprzętowej akceleracji generowania i przetwarzania grafiki na kartach graficznych. Oprócz biblioteki OpenGL, VTK umożliwia wykorzystanie biblioteki Mesa, będącej implementacją OpenGL z otwartym kodem oraz tworzenie obrazów przy pomocy oprogramowania do renderingu zgodnego z programem RenderMan[5].

Wynikiem działania algorytmów mapujących są *dane graficzne* zbudowane z tzw. podstawowych obiektów graficznych (ang. *graphics primitives*). W zależności od implementacji, lista obiektów podstawowych może być różna. Obecna wersja VTK wykorzystuje bibliotekę OpenGL, z następującymi obiektami podstawowymi (rysunek 3.2):

- punkt (ang. *point, vertex*) (`GL_POINTS`) ma zestaw parametrów, takich jak: położenie (współrzędne), kolor oraz wektor normalny,
- linia (ang. *line*) (`GL_LINE`) łączy dwa punkty, tworzy krawędź,
- krzywa (ang. *polyline*) (`GL_LINE_STRIP`) będąca ciągiem połączonych linii,
- krzywa zamknięta (`GL_LINE_LOOP`) zbudowana jest z serii połączonych linii w których ostatni punkt jest połączony z pierwszym,
- trójkąt (ang. *triangle*) (`GL_TRIANGLES`) rozpięty na trzech punktach, ma trzy krawędzie,
- pas trójkątów (ang. *triangle atrip*) (`GL_TRIANGLE_STRIP`) jest zbiorem trójkątów. w których sąsiadujące trójkąty mają wspólne krawędzie,
- wachlarz trójkątów (ang. *triangle fan*) (`GL_TRIANGLE_FAN`) jest zbiorem trójkątów z jednym wspólnym wierzchołkiem,

[5] RenderMan jest profesjonalnym oprogramowaniem, opracowanym przez firmę Pixar, służącym do tworzenia fotorealistycznych obrazów, głównie na potrzeby przemysłu filmowego.

- czworokąt (ang. *quad*) (`GL_QUADS`) rozpięty na czterech punktach, rozmieszczonych na jednej płaszczyźnie,
- pas czworokątów (ang. *quad strip*) (`GL_QUAD_STRIP`) to zbiór czworokątów, którym sąsiadujące czworokąty posiadają wspólną krawędź,
- wielokąt (ang. *polygon*) (`GL_POLYGON`) to zbiór krawędzi tworzących zamknięty obszar, zazwyczaj rozpięty na punktach leżących na jednej płaszczyźnie, trójkąt i czworokąt są przykładami wielokąta.




**Rys. 3.2.** Podstawowe obiekty graficzne biblioteki OpenGL, wykorzystywanej przez VTK do generowania obrazów cyfrowych

Obiekty podstawowe stanowią cegiełki, z których podczas renderingu budowany jest cyfrowy obraz. Współczesne karty graficzne, zgodne ze standardem OpenGL, zapewniają w pełni sprzętową obróbkę obiektów podstawowych. Obejmuje ona obliczenia dotyczące transformowania współrzędnych wierzchołków (skalowanie, przesuwanie czy rotację) jak również wszystkie etapy „rysowania" obiektów podstawowych (więcej informacji na ten temat można znaleźć w rozdziale 5.2). Jednak wydajność przetwarzania obiektów podstawowych dla poszczególnych typów kart graficznych może być znacznie różna. Dzieje się tak ponieważ procesory graficzne, są zwykle optymalizowane do bardzo wydajnego przetwarzania ograniczonej liczby obiektów podstawowych, najczęściej jest to tylko lista trójkątów.

## 3.2.3. Podstawowe obiekty modelu graficznego VTK

Przyjęty w VTK model graficzny wykorzystuje klasy, których nazewnictwo nawiązuje do przemysłu filmowego. Podstawowym pojęciem jest wirtualna *scena,* na której znajdują się *aktorzy,* będący wynikiem działania strumienia wizualizacji, *światła* oraz *kamery.* Widok z wirtualnej kamery prezentowany jest w oknach wizualizacji.

Do najważniejszych klas *modelu graficznego* należą:

**vtkActor, vtkActor2D, vtkVolume** – klasy używane do reprezentowania *danych graficznych* (czyli wyjścia z obiektów mapujących), na wirtualnej scenie.

**vtkLight** – obiekty reprezentujące wirtualne światło, zarówno równoległe (z nieskończoności), jak i punktowe (pozycyjne). Mają takie parametry jak: jasność, kolor, typ, kąt stożka światła (dla punktowego), współczynniki tłumienia wraz z odległością oraz zanikania. Dodatkowo mają macierze transformacji, określające miejsce w przestrzeni, gdzie się znajdują.

**vtkCamera** – wirtualna kamera, służąca do określenia punktu w przestrzeni, z którego widziana jest sceny (ang. *view point*) oraz punktu, na który kamera patrzy (ang. *focal point*). Dostępne metody umożliwiają między innymi: określenie i zmianę współrzędnych kamery, zmianę szerokości kąta patrzenia oraz typu projekcji (ortogonalna lub perspektywiczna).

**vtkProperty, vtkProperty2D** – określa właściwości optyczne powierzchni obiektów graficznych, takie jak: kolory (ogólny, otaczający, rozproszony, połysku, krawędzi), współczynniki modelu oświetlenia (np. odbicia światła ang. *specular power,* rozproszenia ang. *diffuse* oraz współczynnik natężenia światła otoczenia ang. *ambient*), przeźroczystość, reprezentację geometrii obiektu (punkty, krawędzie, powierzchnie) oraz metodę cieniowania (płaskie, Gourauda, Phonga).

**vtkMapper, vtkMapper2D** – są to klasy abstrakcyjne, z których wywodzą się klasy stanowiące pomost między wizualizowanymi danymi a reprezentującymi je w postaci graficznej wirtualnymi aktorami.

**vtkTransform** – obiekty te mogą być wykorzystywane do transformacji afinicznych w trzech wymiarach, opisanych przez macierz przekształceń jednorodnych ($4 \times 4$).

**vtkLookupTable, vtkColorTransferFunction** – obiekty używane do tworzenia tzw. tablic kolorów, służących do mapowania danych skalarnych (indeksów tablicy) do odpowiadających im wartości koloru. Kolor może być kodowany zarówno w postaci triady RGB, z uwzględnieniem kanału przeźroczystości RGBA oraz w systemie HSV.

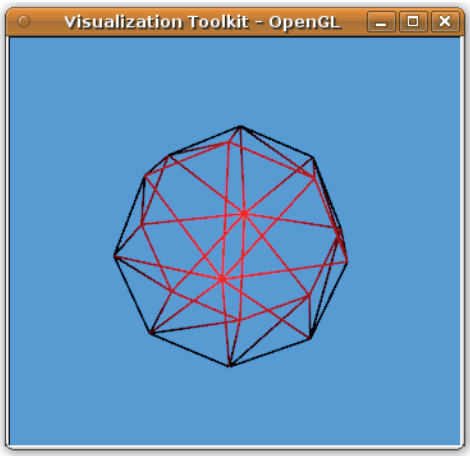
**vtkRenderer** – jest obiektem, kontrolującym proces tworzenia obrazu. Rendering to proces przetwarzania danych geometrycznych, wirtualnego światła i widoku z kamery na cyfrowy obraz. Wykonuje również transformacje między układami współrzędnych: globalnym opisującym przestrzeń sceny, lokalnymi poszczególnych aktorów oraz współrzędnych wyświetlania na ekranie.

**vtkRenderWindow** – abstrakcyjny obiekt, którego implementacje umożliwiają umieszczenie okna z wynikami wizualizacji w interfejsie użytkownika. Metody obiektu umożliwiają synchronizację, zmianę rozmiarów (rozdzielczości) okna wizualizacji, zarządzanie buforowaniem oraz tworzeniem obrazów stereowizyjnych.

**vtkRenderWindowInteractor** – stanowi niezależny od platformy mechanizm interakcji z oknem wizualizacji. Klasa ta, wspólnie z klasą **vtkInteractorStyle** umożliwia interaktywne sterowanie wizualizacją, np. ruchy wykonywane myszką mogą sterować położeniem kamery. W zależności od konkretnej implementacji stosowanej klasy **vtkIn-**

**teractorStyle** źródłem interakcji może być użytkownik (poprzez myszkę, klawiaturę lub inny kontroler), jak również inne obiekty (np. licznik czasu (ang. *timer*) sterujący prędkością animacji).

Kombinacja powyższych obiektów, w połączeniu z wynikami działania strumienia wizualizacji, umożliwia uzyskanie interaktywnej wizualizacji danych wejściowych. Listing 3.1 zawiera przykładowy program w języku C++, którego wynik działania przedstawiono na rysunku 3.3. Listing 3.1 zawiera pełen kod, ale do poprawnej kompilacji niezbędny jest jeszcze plik projektu *CMakeLists.txt*, którego zawartość przedstawiono w listingu 3.2.




**Rys. 3.3.** Wynik działania przykładowego programu

**Listing 3.1:** Przykładowy program VTK (plik sfera.cxx).

```
1  #include <vtkSphereSource.h>
2  #include <vtkPolyData.h>
3  #include <vtkSmartPointer.h>
4  #include <vtkPolyDataMapper.h>
5  #include <vtkActor.h>
6  #include <vtkProperty.h>
7  #include <vtkCamera.h>
8  #include <vtkRenderWindow.h>
9  #include <vtkRenderer.h>
10 #include <vtkRenderWindowInteractor.h>
11
12 int main(int , char *[])
13 {
14  //POCZĄTEK STRUMIENIA WIZUALIZACJI
15   // źródło danych - parametryczna sfera
16   vtkSmartPointer<vtkSphereSource> sphereSource =
17       vtkSmartPointer<vtkSphereSource>::New();
18   sphereSource->SetCenter(0.0, 0.0, 0.0);            // środek sfery
19   sphereSource->SetRadius(5.0);                      // średnica sfery
20   sphereSource->SetThetaResolution(8);               // parametry określające liczbę
```

```
21    sphereSource->SetPhiResolution(4);                    // ścian sfery
22
23    // obiekt mapujący
24    vtkSmartPointer<vtkPolyDataMapper> mapper =
25        vtkSmartPointer<vtkPolyDataMapper>::New();
26    mapper->SetInputConnection( sphereSource->GetOutputPort() );  // podłączenie parametrycznej
27                                                                  // sfery do obiektu mapującego
28
29   //CZĘŚĆ ODPOWIADAJĄCA ZA GRAFICZNĄ REPREZENTACJĘ STRUMIENIA WIZUALIZACJI
30   // aktor reprezentujący dane na wirtualnej scenie
31    vtkSmartPointer<vtkActor> actor =
32        vtkSmartPointer<vtkActor>::New();
33    actor->SetMapper(mapper);                             // przypisanie aktorowi obiektu mapującego
34    actor->GetProperty()->SetColor(1,0,0);                // ustalenie koloru aktora
35    actor->GetProperty()->SetLineWidth(2);                // ustalenie grubości linii
36    actor->GetProperty()->SetRepresentationToWireframe();  // reprezentacja w postaci kratownicy
37
38    // obiekty odpowiedzialne za generowanie obrazu i interakcję
39    vtkSmartPointer<vtkRenderer> renderer =
40        vtkSmartPointer<vtkRenderer>::New();              // wirtualna scena
41    vtkSmartPointer<vtkRenderWindow> renderWindow =
42        vtkSmartPointer<vtkRenderWindow>::New();          // okno programu
43    renderWindow->AddRenderer(renderer);
44    vtkSmartPointer<vtkRenderWindowInteractor> renderWindowInteractor =
45        vtkSmartPointer<vtkRenderWindowInteractor>::New(); // obiekt odpowiedz.za sterowanie
46                                                           // wizualizacją myszką
47    renderWindowInteractor->SetRenderWindow(renderWindow); //podłączenie interaktora do okna
48
49    renderer->AddActor(actor);                            // dodanie aktora do sceny
50    renderer->SetBackground(.3, .6, .9);                  // zmiana koloru tła
51    renderer->GetActiveCamera()->Azimuth(5);              // obrót kamery
52    renderer->GetActiveCamera()->Elevation(15);
53    renderer->ResetCamera();
54
55    // uruchomienie strumienia wizualizacji
56    renderWindow->Render();
57
58    // włączenie interakcji (sterowanie myszką)
59    renderWindowInteractor->Start();
60
61    return EXIT_SUCCESS;
62  }
```

**Listing 3.2:** Plik projektu (CMakeLists.txt) niezbędny przy kompilacji przykładu *sfera.cxx* programem *cmake*

```
1  cmake_minimum_required(VERSION 2.6)
2
3  PROJECT(Sfera)
4
5  FIND_PACKAGE(VTK REQUIRED)
6  INCLUDE(\${VTK_USE_FILE})
7
8  ADD_EXECUTABLE(sfera sfera.cxx)
9
10 TARGET_LINK_LIBRARIES(sfera vtkHybrid)
```

Pierwsze wiersze (od 1 do 10) zawierają dołączenia niezbędnych plików nagłówkowych. Wiersz 16 jest początkiem strumienia wizualizacji, zawiera deklarację źródła danych, obiektu `sphereSource`, w tym przypadku parametrycznej sfery. W kolejnych wierszach ustalany jest środek sfery (18), jej średnica (19) oraz liczba równoleżników i południków (20,21). Wiersz 24 zawiera deklarację obiektu mapującego `mapper`, którego wejściem jest obiekt generujący sferę. Na tym obiekcie kończy się *strumień wizualizacji,* zaczyna zaś część odpowiedzialna za generowanie obrazu, czyli *rendering.* W wierszu 31 powoływany jest do istnienia aktor, który będzie reprezentował dane z „generatora kul" na wirtualnej scenie. W wierszu 33 następuje przypisanie konkretnego *mappera* do aktora, zaś w kolejnych dokonywana są zmiany wyglądu aktora. Odbywa się to poprzez pobranie z aktora jego domyślnych *właściwości* (ang. *property*), a następnie wywołaniu metod odpowiedzialnych za zmianę koloru, grubości linii oraz włączenie wyświetlania aktora w postaci siatkowej (ang. *wireframe*). Aktor wymaga sceny, która powstaje z połączenia dwóch obiektów: `vtkRenderer`, który odpowiada za tworzenia obrazu wizualizacji. Obiekt `renderer` umieszczany jest komendą z wiersza 43. w oknie programu, obiekcie `vtkRenderWindow`. W celu zapewnienia interakcji z programem, w wierszu 44. utworzono tzw. interaktora. Obiekt ten, podłączony do `renderWindow`, umożliwia przejęcie i interpretację komunikatów docierających z klawiatury i myszki do okna programu. Dzięki temu, możliwe będzie, po zainicjalizowaniu rendereingu (wiersz 56) i włączeniu interaktora (wiersz 59), interaktywne sterowanie np. położeniem kamer. W wierszu 49 następuje dodanie aktora do wirtualnej sceny. Dokonywana jest również zmiana koloru tła oraz obrót kamery w osi poziomej o 5 stopni i 15 w osi pionowej. Metoda `ResetCamera()` powoduje obliczenie nowych współczynników transformacji dla kamery.

### 3.2.4. Zarządzanie pamięcią

Komentarza wymaga jeszcze sposób tworzenia obiektów, ponieważ VTK nie obsługuje „klasycznych" metod tworzenia `new()` oraz usuwania obiektów `delete()`. Biblioteka VTK wykorzystuje mechanizmy automatycznego zwalniania nieużywanych zasobów (ang. *garbage collection*). Każdy obiekt ma licznik referencji, na podstawie którego można określić, kiedy zajmowany przez niego obszar pamięci można bezpiecznie zwolnić. Licznik obiektu jest zwiększany, gdy jest używany przez inny obiekt. Jeżeli zostanie od niego odłączony, jego licznik referencyjny zostaje zmniejszony o 1. Gdy licznik referencji osiągnie wartość 0, obiekt jest automatycznie usuwany z pamięci. Niestety metoda ta nie gwarantuje braku „wycieków pamięci" – w pewnych sytuacjach (np. przy rekurencyjnych odwołaniach między obiektami) pamięć nie jest poprawnie zwalniana.

Podstawową metodą tworzenia obiektów VTK w języku C++, jest używanie metod `New()` oraz `Delete()`, w następujący sposób:

- tworzenie obiektu: `vtkActor actor = vtkActor::New()` – tworzony jest wskaźnik oraz przydzielana jest pamięć dla obiektu, wywoływany jest konstruktor klasy będący metodą chronioną (ang. *protected method*),
- usuwanie obiektu: `actor->Delete()` – obiekt jest „zgłaszany do usunięcia", pamięć zostaje zwolniona wówczas, gdy obiekt nie jest używany przez inne obiekty w programie, metoda ta musi się pojawić w kodzie programu!

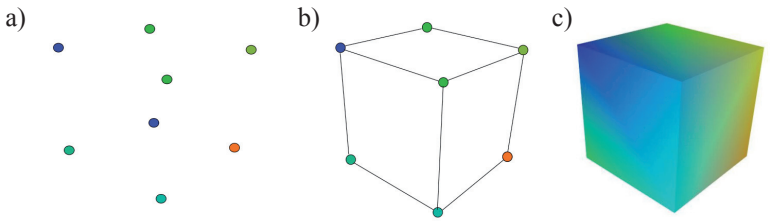
Inną metodą zarządzania pamięcią jest wykorzystanie mechanizmu tzw. *sprytnych wskaźników,* zaimplementowanych w klasie `vtkSmartPointer`. Szczegółowe wyjaśnienie działania sprytnych wskaźników, wraz z przykładami, znajduje się na stronie `http://www.itk.org/Wiki/VTK/Tutorials/SmartPointers`. Wszystkie obiekty VTK występujące w przykładowym programie (listing 3.1) są obiektami dynamicznymi, które powstały właśnie poprzez obiekt `vtkSmartPointer`. Obiekt ten tworzy wskaźniki konkretnego typu (używając mechanizmu szablonów, ang. *template*), a następnie zarządza przydziałem oraz zwalnianiem pamięci obiektu w sposób dynamiczny, na podstawie zliczania odniesień. Jeżeli utworzony obiekt nie jest już niegdzie wykorzystywany, wówczas jest automatycznie usuwany z pamięci. W tym podejściu nie używana jest bezpośrednio w kodzie metoda `Delete()`. Stworzenie nowego obiektu obywa się w następujący sposób: `vtkSmartPointer<vtkActor> actor = vtkSmartPointer <vtkActor>::New()`.

Mechanizm *sprytnych wskaźników* jest zdecydowanie bardziej „szczelny" niż metoda wcześniej opisana i jest szczególnie zalecany przy tworzeniu bezpiecznych programów.

## 3.2.5. Reprezentacja danych

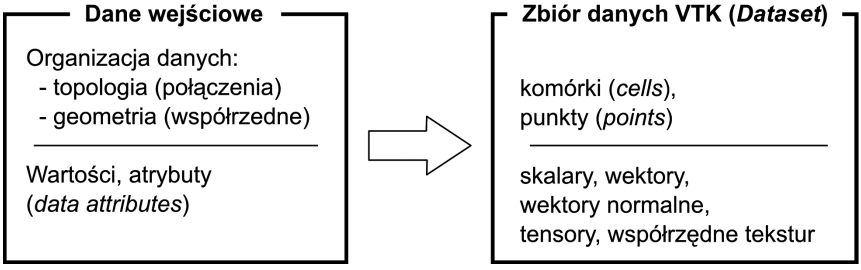
Postrzegany przez nas rzeczywisty świat jest zazwyczaj ciągły, zaś dane, szczególnie w postaci cyfrowej, mają charakter dyskretny. Dane opisujące zjawiska czy też obiekty posiadają *atrybuty*, czyli charakteryzujące je wartości. Wizualizacja ciągłych zjawisk, opisanych dyskretnymi wartościami, wymaga zastosowania *interpolacji.* Do interpolacji wykorzystuje się *topologię*, czyli informację o tym, które dane i w jaki sposób są ze sobą połączone. Topologia danych opisuje połączenia niezależnie od przeprowadzanych przekształceń geometrycznych. Dzięki zdefiniowanej topologii możliwe jest wyznaczenie wartości pośrednich. Zmiana topologi połączeń prowadzi do zmian w sposobie interpolacji danych, a tym samym zmienia efekt wizualizacji.

Oprócz topologii, dane zawierają również *informacje geometryczne* o ich rozmieszczeniu w przestrzeni. Rysunek 3.4 przedstawia przykład danych. Jest to osiem punktów w przestrzeni, reprezentujących np. zmierzoną temperaturę. Na rysunku 3.4a punkty zostały rozmieszczone zgodnie z przypisanymi im wartościami geometrycznymi (współrzędnymi), zaś kolor odpowiada wartości atrybutu, w tym przypadku temperaturze. Rysunek 3.4b przedstawia te same dane, jednak po uwzględnieniu topologii, zaś 3.4c wynik interpolacji.




**Rys. 3.4.** Przykład zastosowania topologii: a) dane bez informacji o połączeniach, b) uwzględnienie topologii, c) interpolacja danych

Zastosowany w VTK *model danych* (ang. *dataset*) przyjmuje, że dowolnym, abstrakcyjnym danym, można przypisać dwa rodzaje informacji: *wartości* oraz skojarzoną z nimi *strukturę danych.* Model ten przedstawiony jest na rysunku 3.5. Struktura lub inaczej organizacja danych opisana jest przez *topologię* (połączenia) oraz *geometrię* (współrzędne). Do przechowywania informacji o strukturze danych VTK używa licznej rodziny obiektów tzw. *komórek* (ang. *cells*). Komórki stanowią pomost pomiędzy danymi dyskretnymi a ich interpolacją. Dla danych bez topologii informacje przechowywane są w punktach. Z kolei *wartości* (ang. *data attributes*) mogą mieć różną reprezentacją, mogą to być: wartości skalarne, wektorowe, tensorowe, tekstury, itp.




**Rys. 3.5.** Dane wejściowe i odpowiadający im model danych VTK (ang. *dataset*)

Przyjęty model danych, w którym wyróżnia się: wartości, topologię oraz geometrię, reprezentowane przez zbiór punktów (ang. *points*) i komórek (ang. *cells*), jest więc odzwierciedleniem dyskretnej natury danych. Punkty określają jedynie miejsce, w których znamy wartość ciągłych zjawisk, zaś komórki umożliwiają dokonanie interpolacji między punktami.

## 3.2.6. Typy komórek vtkCell

Komórki definiowane są poprzez określenie *typu*, a następnie przypisanie uporządkowanej listy punktów, tzw. *listy połączeń* (ang. *connectivity list*). Jeżeli symbolem $C_i$ oznaczymy komórkę, wówczas listę punków opisujących komórkę można zapisać w następujący sposób: $C_i = \{p_1,p_2,p_3,...,p_n\}$, gdzie $p_i \in P$, zaś $P$ jest zestawem $n$-wymiarowych punktów. Liczba punktów $n$ określa *rozmiar* komórki. Komórka $C_i$ „używa" punktu $p_i$, gdy $p_i \in C_i$. Definiuje się również funkcję wykorzystania punktu tzw. „używalności punktu" $U(p_i)$, która zwraca zbiór wszystkich komórek używających punktu $p_i$:

$$U(p_i) = \{C_i : p_i \in C_i\} \tag{3.1}$$

Przekształcenie to umożliwia, między innymi, „poruszanie się" między sąsiadującymi ze sobą komórkami.

Biblioteka VTK ma zaimplementowanych wiele typów komórek [6]. Podstawowym kryterium podziału jest *wymiar topologiczny komórki.* Określa on najmniejszą liczbę potrzebnych

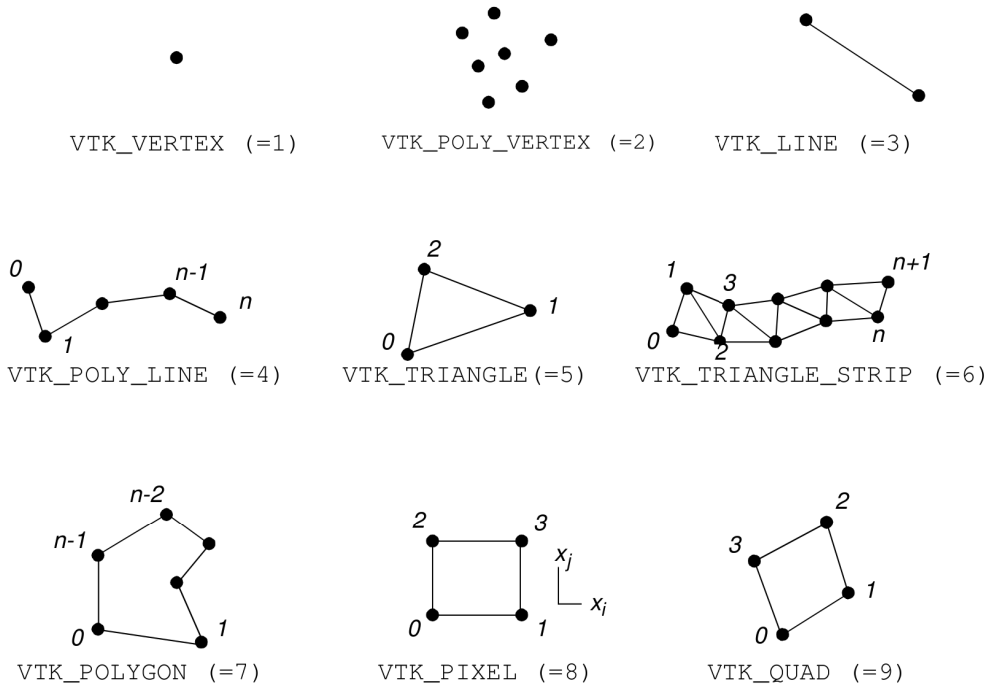
[6] `http ://www.vtk.org/doc/release/5.6/html/a00262.html`

do opisania współrzędnych punktu w przestrzeni *N*-wymiarowej: punkty mają wymiar 0, a krzywe wymiar 1 (tylko jedna liczba wystarczy, by określić, gdzie się znajduje dowolny punkt na krzywej), powierzchnie mają wymiar topologiczny równy 2, zaś bryły geometryczne wymiar topologiczny 3. Innym podziałem może być zastosowana funkcja interpolująca, która może być: liniowa lub nieliniowa.

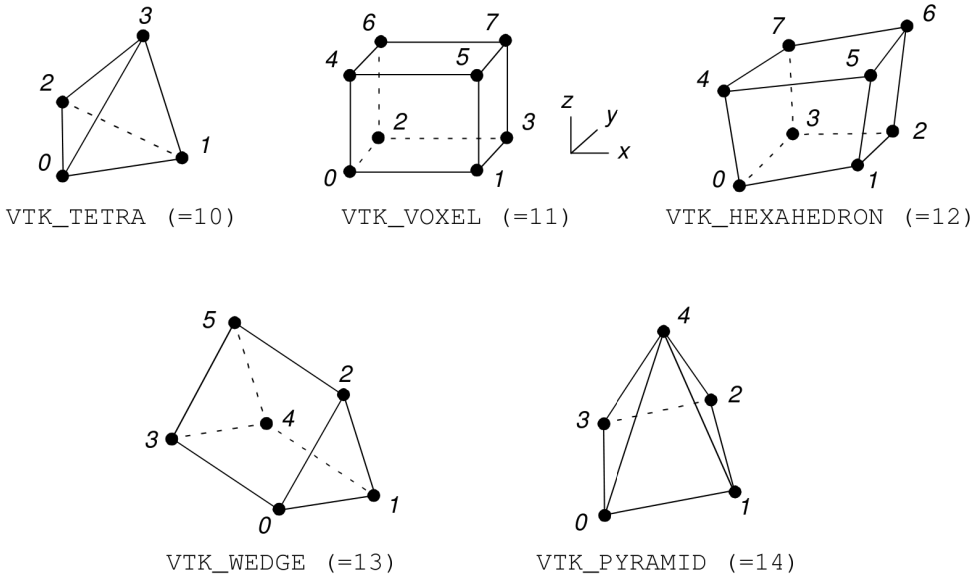
Komórki definiowane są poprzez: *typ* oraz posortowaną *listę połączeń* (ang. *connectivity list*). Lista połączeń zawiera indeksy punktów, te zaś prowadzą do listy punktów zawierającej ich współrzędne geometryczne.

Rysunki 3.6 oraz 3.7 przedstawiają zaimplementowane typy komórki z liniową interpolacją, o następujących wymiarach topologicznych:

- o zerowym wymiarze (0D):
  - wierzchołek `VTK_VERTEX` jest najprostszym i podstawowym typem komórki, wyznacza jeden punkt w przestrzeni, niesie informację jedynie o położeniu geometrycznym bez definiowania połączeń;
  - zbiór wierzchołków `VTK_POLY_VERTEX` definiowany jest poprzez listę punktów w przestrzeni;




**Rys. 3.6.** Przykłady komórek o zerowych wymiarach 0D topologicznych (1,2), 1D (3,4) oraz 2D (5–9), nazwy i oznaczenia na podstawie vtkCell



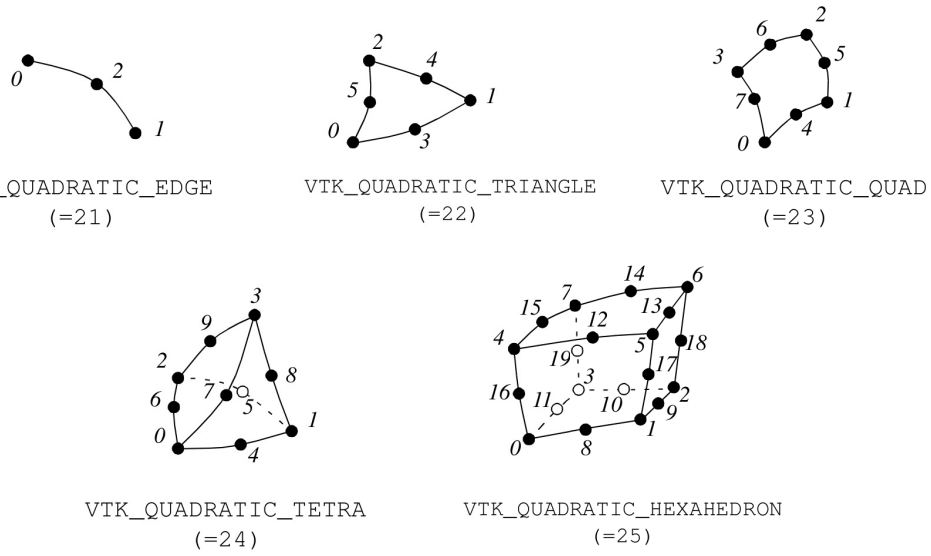

**Rys. 3.7.** Przykłady komórek o trójwymiarowej topologii

- jednowymiarowe (1D):
  - linia `VTK_LINE` są podstawowym typem jednowymiarowym, definiowanym przez dwa punkty, przy czym kierunek wyznaczany jest od punktu pierwszego do drugiego;
  - krzywa `VTK_POLY_LINE` jest zbiorem linii, definiowanym przez uporządkowaną listę $n + 1$ punktów, gdzie $n$ jest liczbą linii tworzących krzywą, pary punktów $(i, i + 1)$ wyznaczają poszczególne linie;
- dwuwymiarowe (2D):
  - trójkąt `VTK_TRIANGLE` jest podstawowym obiektem o dwuwymiarowej topologii, definiowanym przez listę trzech punktów, kolejność punktów wyznacza kierunek wektora normalnego (zasada prawej dłoni – jeżeli palce wskazują kierunek wzrostu indeksów punktów, to odchylony kciuk wyznacza kierunek wektora normalnego);
  - pas trójkątów `VTK_TRIANGLE_STRIP` jest zbiorem połączonych trójkątów, wyznaczanym przez uporządkowaną listę $n + 2$ punktów, gdzie $n$ jest liczbą trójkątów, pojedynczy trójkąt definiują trzy kolejne punkty listy połączeń: $(i, i + 1, i + 2)$, gdzie $0 < i < n$, punkty nie muszą leżeć na jednej płaszczyźnie;
  - czworokąt `VTK_QUAD` wyznaczany jest przez uporządkowaną listę czterech punktów, leżących na jednej płaszczyźnie i tworzących figurę wypukłą, o nieprzecinających się krawędziach; wektor normalny wyznaczany jest podobnie jak w przypadku trójkąta, poprzez kolejność punktów (przeciwną do kierunku ruchu wskazówek zegara) i zgodnie z regułą prawej dłoni;

  - piksel `VTK_PIXEL` jest wyznaczany przez cztery punkty położone na jednej płaszczyźnie i tworzące prostokąt, którego równoległe krawędzie pokrywają się z wybranymi osiami układu współrzędnych, dodatkowo punkty sortowane są w taki sposób, by ich indeksy rosły wraz ze wzrostem współrzędnych układu odniesienia; komórka tego typu stanowi zoptymalizowaną obliczeniowo wersję czworokąta; należy zwrócić uwagę na nazwę „piksel", która zazwyczaj definiowana jest trochę inaczej, jako *pojedynczy element obrazu*; w przypadku komórki typu *piksel* zawiera ona cztery „elementy obrazu";
  - wielokąt `VTK_POLYGON` jest zamkniętą figurą, definiowaną przez uporządkowaną listę więcej niż trzech punktów leżących na jednej płaszczyźnie, wielokąt nie musi być wypukły, nie mogą jednak występować w nim wewnętrzne zapętlenia oraz przecięcia krawędzi; wielokąt zawiera *n* krawędzi, gdzie *n* to liczba wierzchołków; wektor normalny wyznaczony jest w sposób bezpośredni, na podstawie odwrotnej do ruchu wskazówek zegara kolejności punktów i zasadzie prawej dłoni,
- trójwymiarowe (3D):
  - czworościan `VTK_TETRA` definiuje lista czterech punktów, które nie leżą na jednej płaszczyźnie; czworobok posiada sześć krawędzi oraz cztery wierzchołki;
  - sześciościan `VTK_HEXAHEDRON` definiuje lista ośmiu punktów, nieleżących na jednej płaszczyźnie; zbudowany jest z sześciu czworokątnych ścian, dwunastu krawędzi oraz ośmiu wierzchołków, niedozwolone są przecięcia ścian i krawędzi, sześciościan musi być wypukły;
  - woksel `VTK_VOXEL` jest optymalizowaną obliczeniowo wersją sześciościanu, pokrywającą się z nim topologicznie, jednak z narzuconymi ograniczeniami geometrycznymi: wszystkie krawędzie oraz ściany muszą być prostopadłe lub równoległe do osi układu współrzędnych; dodatkowo, lista punktów musi być posortowana tak, by indeksy punktów wzrastały wraz ze wzrostem wartości współrzędnych układu odniesienia (rysunek 3.7); podobnie jak w przypadku typu piksel, ten typ komórki definiuje osiem „pojedynczych elementów" obrazu przestrzennego (ang. *volume data*), które są rozmieszczone „w rogach" komórki typu *woksel*;
  - klin `VTK_WEDGE` jest komórką definiowaną listą sześciu punktów połączonych dziewięcioma krawędziami, które tworzą dwie ściany trójkątne oraz trzy czworokątne, niedozwolone są przecięcia krawędzi i ścian, figura musi być wypukła,
  - piramida `VTK_PYRAMID` definiowana jest przez listę pięciu punktów, pierwsze cztery leżą na jednej płaszczyźnie i tworzą czworokątną ścianę, piąty zaś wyznacza z pozostałymi cztery ścianki trójkątne (rysunek 3.7).

Komórki jednowymiarowe stosowane są zazwyczaj do oznaczania konturów i krawędzi. Piksele, ze względu na dużą wydajność wynikającą z prostej budowy, stanowią bazę do przechowywania cyfrowych obrazów (zwykle regularnych macierzy danych dwuwymiarowych), natomiast woksele są wykorzystywane do reprezentacji danych medycznych (trójwymiarowych macierzy, wyników np. z tomografii komputerowej). Nieregularne komórki dwuwymiarowe (trójkąt, czworokąt) oraz trójwymiarowe (czworościan i sześciościan) są szczególnie chętnie stosowane w symulacjach komputerowych, zwłaszcza w tzw. *metodach*

*elementów skończonych* (ang. *FEM, Finite-Element Method*), mechanice płynów i innych. Przedstawione typy komórek umożliwiają wykonanie jedynie interpolacji liniowych. Może się jednak zdarzyć, że dokładność takich obliczeń jest niewystarczająca. Wówczas stosuje się funkcje nieliniowe, np. funkcje kwadratowe, B-spline czy też NURBS[7]. VTK ma tylko jeden typ *nieliniowych komórek*, które umożliwiają interpolację kwadratową. Wymaga ona dodatkowego, trzeciego punktu umieszczanego na każdej krawędzi. Na rysunku 3.8 przedstawiono zaimplementowane nieliniowe komórki.




**Rys. 3.8.** Przykłady komórek, które umożliwiają kwadratową interpolację danych

Główną różnicą między komórkami liniowymi i nieliniowymi jest sposób, w jaki są wizualizowane i przetwarzane przez filtry. Najprostszą metodą wizualizacji jest zamiana komórek nieliniowych na większą liczbę komórek liniowych (ang. *tessellation*). Tak „zagęszczone" dane są następnie przetwarzane i wizualizowane przy pomocy „standardowych" liniowych metod. Innym podejściem jest opracowanie filtrów i algorytmów, które działają bezpośrednio na komórkach nieliniowych. Ze względu na duże skomplikowanie problemu (np. różne „klasy" nieliniowości wymagają zupełnie innej implementacji) VTK nie wspiera w ten sposób wielu funkcji nieliniowych. Sama wizualizacja „nieliniowych danych" (opisanych przez nieliniowe komórki) może być również wykonana przy wsparciu bibliotek graficznych. Prawdopodobnie, wraz ze wzrostem mocy obliczeniowych kart graficznych, udostępnione zostaną wydajne sprzętowe metody wizualizacji danych opisanych krzywymi parametrycznymi.

[7] Przegląd zagadnień związanych z nieliniową interpolacją danych: `http://en.wikipedia.org/wiki/Spline_interpolation`

### 3.2.7. Typy atrybutów

*Wartości* lub też inaczej *atrybuty* (ang. *attribute data*), są informacjami powiązanymi ze strukturą danych: z geometrią i topologią. Wartości mogą być przypisane zarówno do punktów, jak i do komórek oraz ich części (krawędzi i ścianek). Przykładem atrybutów mogą być wartości temperatury (skalar) lub prędkości (wektor) przypisane do punktów, masa przypisana do komórki lub strumień ciepła przypisany do ścianki komórki.

Biblioteka VTK dzieli atrybuty danych na następujące typy:

- skalary (ang. *scalars*) – dane mające tylko jedną wartość w każdym punkcie, jest to najczęściej spotykany typ danych; przykładem może być: temperatura, ciśnienie, gęstość czy cena;
- wektory (ang. *vectors*) – dane mające: wartość, kierunek oraz zwrot; w przestrzeni dwuwymiarowej określone przez dwie składowe ($u$, $v$), zaś w trójwymiarowej przez trzy ($u$, $v$, $w$), przykładem jest prędkość lub gradient temperatur;
- wektory normalne (ang. *normals*) – wektor jednostkowy o długości równej 1; stosowane głównie do obliczeń kolorów powierzchni oświetlonych oraz określenia ich orientacji przestrzennej;
- współrzędne tekstur (ang. *texture coordinates*) – służą do mapowania punktu z kartezjańskiego układu współrzędnych do przestrzeni 1, 2 lub 3 wymiarowej tekstury; tekstura jest regularną siatką (zazwyczaj obrazem), zawierającą informacje o kolorze, intensywności, często również o przeźroczystości; współrzędne tekstur umożliwiają wizualizację np. globusa, poprzez nałożenie dwuwymiarowej grafiki prezentującej mapę ziemi na trójwymiarową kulę;
- tensory (ang. *tensors*) – stanowią matematyczne uogólnienie wektora i macierzy; wielkości tensorowe reprezentuje się w postaci macierzy kwadratowej, tensor rzędu $k$ jest k-wymiarową macierzą kwadratową; tensor rzędu zerowego to skalar, rzędu pierwszego to wektor zaś drugiego to macierz; przykładami wielkości tensorowych są tensory naprężeń i odkształceń mechanicznych.

### 3.2.8. Implementacja

Obiektem VTK o szczególnym znaczeniu jest abstrakcyjna superklasa `vtkAbstractArray`, która stanowi protoplastę „abstrakcyjnego pojemnika" na dowolne dane, przechowywane w pamięci w postaci obszaru ciągłego. W obiektach wywodzących się z tej klasy przechowywane są wszelkiego rodzaju informacje zarówno *atrybuty* (wartości) danych, jak i dane dotyczące *topologii* (np. listy połączeń) i *geometrii* (np. współrzędne). Z tej abstrakcyjnej klasy wywodzą się dwie klasy służące do przechowywania danych tekstowych `vtkStringArray` i `vtkUnicodeStringArray` oraz klasa `vtkDataArray`.

`vtkDataArray` jest nadklasą dla *danych numerycznych* przechowywanych w postaci tablic danych i stanowi podstawowe narzędzie do przechowywania i zarządzania tymi danymi. Z klasy tej wywodzi się duża grupa obiektów VTK[8], które są wyspecjalizowane do

[8] Pełna lista znajduje się w dokumentacji online Doxygen klasy `vtkDataArray`

obsługi danych w tzw. *formatach podstawowych* (`bit`, `char`, `int`, `uint`, `float` itp.). Klasy te często mają zaimplementowane metody zoptymalizowane dla konkretnych typów danych, np. wstawianie, dodawanie i kasowanie danych, szybkie adresowanie czy też kopiowania itp. Zarządzanie pamięcią odbywa się w sposób dynamiczny. Obiekty przechowują *wskaźnik do adresu tablicy* danych konkretnego typu (np. int, float), aktualny rozmiar zarezerwowanej i wykorzystanej pamięci oraz wskaźnik do adresu tablicy znaków przechowującą tekstową *nazwę* danych.

Jak widać, w pakiecie VTK zastosowano liniową organizację danych (wskaźnik do adresu tablicy danych). Nie jest ona wygodna, gdy niezbędne jest przechowywanie struktur bardziej skomplikowanych, takich jak np. cztery liczby całkowite opisujące kolor w modelu RGBA lub trzy liczby zmiennoprzecinkowe opisujące współrzędne punktu w trójwymiarowej przestrzeni. W celu ułatwienia dostępu do przechowywanych danych jednego typu, tworzących mniejsze grupy, zaimplementowano mechanizm oparty o tzw. *krotki* (ang. *tuple*)[9]. Ciągła przestrzeń tablicy jest dzielona na mniejsze podciągi (krotki) o zadanej liczbie elementów (ang. *components*). Rozmiar krotki określany jest metodą `SetNumberOfComponents()`, zaś liczba krotek w tablicy danych metodą `SetNumberOfTuples()`. Dla wartości skalarnych krotka ma rozmiar 1, dla wektora trójwymiarowego 3, zaś dla tensora trzeciego rzędu 9. Rozmiar krotki jest stały dla wszystkich wartości w jednej tablicy. Przykłady użycia obiektów tego typu znajdują się w listingu 3.4.

### 3.2.9. Typy zbiorów danych

Przyjęty w VTK model danych łączy organizację danych z jej wartościami. Ze względu na różną organizację danych, wyróżnić można kilka podstawowych *typów zbiorów danych* (rysunek 3.9):

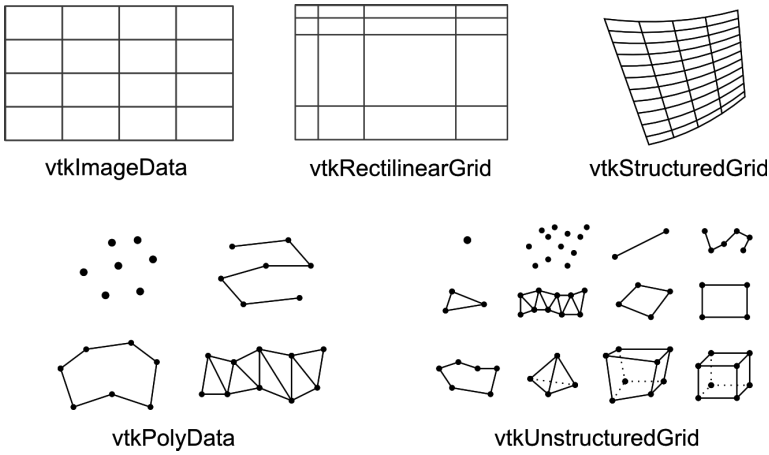
**Image Data** – dane, wyróżniające się równomiernym rozmieszczeniem punktów i komórek na regularnej siatce. Dane zorganizowane są w tabele: wiersze, kolumny oraz poziomy (warstwy), które są równoległe z układem współrzędnych. Dane tego typu mogą zawierać jeden wiersz danych (jednowymiarową linię elementów, np. paletę kolorów), piksele (jak np. typowy obraz dwuwymiarowy) oraz woksele (trójwymiarowe dane medyczne, np. z tomografii komputerowej). Ponieważ dane są zorganizowane regularnie zarówno topologicznie, jak i geometrycznie, współrzędne poszczególnych danych można wyznaczyć w sposób pośredni, co zmniejsza zapotrzebowanie na pamięć. Do wyznaczenia współrzędnych, niezbędna jest informacja o *rozmiarach danych* (ang. *data dimension*) (w przypadku obrazu dwuwymiarowego jest to rozdzielczość), *odległość między danymi* (ang. *data spacing*) *oraz punkt początkowy* (ang. *origin point*).

Rozmiary danych przechowywane są w postaci trójelementowego wektora ($n_x$, $n_y$, $n_z$) który zawiera liczbę punktów w poszczególnych kierunkach (wymiarach). Punkt początkowy (ang. *origin point*) to wektor wyznaczający współrzędne pierwszego punktu danych, określone w trójwymiarowej przestrzeni sceny Odległość między danymi wyznacza, rozmiary pojedynczej komórki. Liczba punktów w danym kierunku pomnożo-

[9] VTK dopuszcza tylko jeden typ danych przechowywanych w krotce.

na, przez odległość między punktami (ang. *spacing*) wyznacza rozmiar geometryczny danych.

Obiekt VTK obsługujący te dane to `vtkImageData`. Podstawowym zastosowaniem tego typu danych jest przechowywanie regularnych macierzy (np. danych obrazowych), w tym danych medycznych, takich jak: obrazy z tomografii komputerowej czy rezonansu magnetycznego.




**Rys. 3.9.** Podstawowe typy danych (ang. *types of dataset*) oraz obiekty biblioteki VTK umożliwiające ich obsługę

**Rectlinear Grid** – dane zorganizowane regularnie topologiczne, podobnie do danych typu *Image Data,* jednak z jedynie częściowo regularną geometrią. Odległość między poszczególnymi wierszami, kolumnami lub warstwami mogą być różne. Definiowane są w postaci list, oddzielnych dla poszczególnych osi współrzędnych.

Obiekt obsługujący ten typ danych to `vtkRectilinearGrid`. Dane tego typu mogą być „specjalizowaną" odmianą danych obrazowych, spotykaną np. w danych medycznych, w których pewne fragmenty są np. skanowane z większą rozdzielczością przestrzenną.

**Structured Grid** – dane zorganizowane są w regularną topologię (siatkę) i nieregularną geometrię. Topologia, wyznaczana jest pośrednio, przez określeniewymiaru danych ($n_x$, $n_y$, $n_z$). Dzięki regularnej topologii, topologiczne współrzędne dowolnego punktu można łatwo wyrazić w postaci indeksów ($i$, $j$, $k$): wiersza, kolumny i warstwy. Współrzędne geometryczne każdego punktu wyznaczane są bezpośrednio, na podstawie tablicy o takich samych wymiarach co dane ($n_x$, $n_y$, $n_z$). Tablica ta zawiera współrzędne poszczególnych punktów danych.

Obiekt VTK obsługujący te dane to `vtkStructuredGrid`. Dane tego typu są szczególnie użyteczne w symulacjach numerycznych, zwłaszcza w mechanice płynów, symula-

cji przesyłu ciepła oraz innych obliczeniach wykorzystujących np. metodę elementów skończonych.

**Polygonal Data** – są to dane, które mogą być bezpośrednio wyświetlone przez kartę graficzną, tzw. podstawowe obiekty graficzne (patrz rysunek 3.2). Do danych tych zaliczają się: punkty, zbiory punktów, linie, krzywe, wielokąty oraz paski trójkątów.

Obiekt `vtkPolyData` umożliwia obsługę tego typu danych. Głównym ich zastosowaniem jest modelowanie obiektów przestrzennych.

**Unstructured Grid** – są to dane o dowolnej organizacji, zarówno topologia, jak i geometria punktów danych przechowywana jest w sposób bezpośredni, w postaci jawnych obiektów. Dowolne dane można zapisać w tym typie, jednak wiąże się to ze znacznym zapotrzebowaniem na pamięć oraz znacznymi kosztami obliczeniowymi.

Obiekt `vtkUnstructuredGrid` jest implementacją danych o dowolnej organizacji. Teoretycznie, format ten może służyć do przechowywania dowolnych danych, jednak ze względu na małą wydajność oraz duże zapotrzebowanie na pamięć korzysta się z niego jedynie w ostateczności.

**Unstructured Points** – dane w postaci jedynie punktów, bez topologii oraz z nieregularną geometrią. Jest to prosty w budowie typ danych, jednak dość znaczący. Często zdarza się, że dane tego typu są danymi wejściowymi, np. zbiorem wartości uzyskanych z „punktów pomiarowych" a celem jest wizualizacja tych danych. Wówczas na drodze obliczeń, w strumieniu wizualizacji, wyznaczana jest „nowa" topologia danych.

Tego typu dane mogą być przechowywane zarówno w vtkPolyData, jak i w vtkUnstructuredGrid.

## 3.3. Proceduralne tworzenie danych

W tym rozdziale zostaną przedstawione przykłady tworzenia podstawowych reprezentacji danych. Zostały one zaczerpnięte z książki *The Visualization Toolkit* [SML04]. Więcej przykładów można znaleźć na stronie `www.vtk.org/Wiki/VTK/Examples`.

### Image Data

Listing 3.3 przedstawia fragment kodu programu, którego zadaniem jest wygenerowanie trójwymiarowych danych typu *Image Data.* Generowane dane są objętościowe i opisują po 26 punktów w każdej osi. Do punktów tych przypisane są wartości skalarne, w postaci liczb typu zmiennoprzecinkowego (*float*). Wartości skalarne opisane są funkcją parametrycznej sfery $F(x,y,z) = (x^2 + y^2 + z^2) - R^2$, gdzie $R$ jest średnicą równą 0,4.

**Listing 3.3:** Przykład tworzenia danych typu Image Data

```
1  vtkImageData *vol = vtkImageData::New(); // deklaracja obiektu przechowującego dane
2  // wyznaczenie TOPOLOGII i GEOMETRII danych:
3    vol->SetDimensions(26,26,26);          // określenie wymiarów tablicy wartości
```

```
4    vol->SetOrigin(-0.5,-0.5,-0.5);        // ustalenie wsp. pierwszego punktu tablicy wartości
5    sp = 1.0/25.0;                         // obliczenie odległości między punktami
6    vol->SetSpacing(sp,sp,sp);             // przypisanie odległości
7  // wyznaczenie WARTOŚCI danych:
8  vtkFloatArray *scalars = vtkFloatArray::New();  // deklaracja tablicy na wartości
9  for (k=0; k<26; k++) {                 // wypełnienie tablicy danymi
10   z = -0.5 + k*sp;
11   kOffset = k*26*26;
12   for (j=0; j<26; j++) {
13      y = -0.5 + j*sp;
14      jOffset = j*26;
15      for (i=0; i<26; i++) {
16         x = -0.5 + i*sp;
17         s = x*x + y*y + z*z - (0.4*0.4);  // wyznaczenie WARTOŚCI danych
18          offset = i + jOffset + kOffset;  // wyznaczenie pozycji WARTOŚCI w tablicy
19          scalars->InsertTuple1(offset,s); // wpisanie wartości do tablicy
20      }
21   }
22 }
23
24 // przypisanie wartości tymczasowych do obiektu przechowującego topologię oraz wartości:
25 vol->GetPointData()->SetScalars(scalars);
26 scalars->Delete();                        // usunięcie danych tymczasowych
27 ...
```

Tworzenie danych typu *Image Data* nie jest szczególnie trudne, ponieważ zarówno topologia, jak i geometria określane są w sposób pośredni (ang. *implicity defined*), na podstawie następujących parametrów: wymiarów (wiersz 3), współrzędnych pierwszego punktu danych (wiersz 4) oraz odległości między punktami (wiersz 6). Wszystkie wartości współrzędnych wyznaczane są względem globalnego układu odniesienia.

Skalarne wartości punktów definiowane są w sposób bezpośredni (ang. *explicity definition*), w postaci tablicy danych (`vtkDataArray`). W tym przypadku jest to tablica liczb zmiennoprzecinkowych (wiersz 8), która jest „wypełniana" danymi (wiersz 19).

Obydwa obiekty w przedstawionym przykładzie tworzone są „starą" metodą (patrz rozdział 3.2.4, przez zadeklarowanie wskaźnika oraz wywołanie metody `New()`. Metoda ta, jak już wcześniej zostało wyjaśnione, nie zapobiega wyciekom pamięci, jednak została przytoczona ze względu na swoją zwięzłość i tym samym lepszą czytelność. Należy zwrócić uwagę na wiersz 26, w której wywoływana jest metoda `Delete()` dla obiektu tymczasowego. Obiekt `scalars` nie zostanie usunięty z pamięci natychmiast, ponieważ został on „podłączony" w wierszu 25 do obiektu vol. Tak długo, jak obiekt `vol` będzie korzystał z danych `scalars`, pamięć nie zostanie zwolniona.

Tworzenie danych typu Image Data można więc podzielić na trzy etapy:

1. Zdefiniowanie topologii, przez wyznaczenie rozmiarów metodą `SetDimensions()`.
2. Zdefiniowanie geometrii, przez określenie odległości `SetSpacing()` oraz współrzędnych pierwszego punktu `SetOrigin()`.
3. Stworzenie i wyliczenie wartości (atrybutów) dla wszystkich punktów, których liczba wynika z rozmiarów topologicznych.

### Structured Grid Data

Ten rodzaj danych jest podobny do danych „obrazowych" (Image Data) z tą różnicą, że geometria opisana jest bezpośrednio (ang. *explicity definition*), czyli każdy punkt ma przypisane współrzędne. Kolejny zaprezentowany listing 3.4 przedstawia fragment kodu, który generuje dane wektorowe, zapisane w cylindrycznym układzie współrzędnych: $x = r_i cos(\theta)$, $y = r_i sin(\theta)$, $z = z_i$.

**Listing 3.4:** Przykład tworzenia danych typu Structured Grid

```
1  vtkStructuredGrid *sgrid = vtkStructuredGrid ::New();
2    sgrid->SetDimensions(dims[0],dims[1],dims[2]);// ustalenie rozmiarów siatki (TOPOLOGIA)
3  vtkPoints *points = vtkPointsNew();             // współrzędne punków siatki (GEOMETRIA)
4    points->Allocate(dims[0]*dims[1]*dims[2]);
5  vtkFloatArray *vectors = vtkFloatArray:New();   // wartości wektorowe (ATRYBUTY danych)
6    vectors->SetNumberOfComponents(3);            // wektor 3-wymiarowy
7    vectors->SetNumberOfTuples(dims[0]*dims[1]*dims[2]);  // liczba wektorów równa liczbie punktów
8  // obliczenie współrzędnych (geometrii) oraz wartości
9  deltaZ = 2.0 / (dims[2]-1);
10 deltaRad = (rMax-rMin) / (dims[1]-1);
11 v[2] = 0.0;                                     // trzecia składowa wektora
12 for ( k=0; k<dims[2]; k++ ) {
13   x[2] = -1.0 + k*deltaZ;
14   kOffset = k*dims[0]*dims[1];
15   for ( j=0; j<dims[1]; j++ ) {
16     radius = rMin + j*deltaRad;
17     jOffset = j*dims[0];
18     for ( i=0; i<dims[0]; i++ ) {
19       theta = i*15.0*math.DegreesToRadians();
20       x[0] = radius * cos(theta);               // pierwsza współrzędna punktu
21       x[1] = radius * sin(theta);               // druga współrzędna punktu
22       v[0] = -x[1];                             // pierwsza składowa wektora
23       v[1] = x[0];                              // druga składowa wektora
24       offset = i + jOffset + kOffset;
25          points->InsertPoint(offset,x);
26          vectors->InsertTuple(offset,v);
27     }
28   }
29 }
30 sgrid->SetPoints(points);                       // przypisanie współrzędnych
31   points->Delete();                            // zwolnienie pamięci
32 sgrid->GetPointData()->SetVectors(vectors);     // przypisanie wartości
33   vectors->Delete();                           // zwolnienie pamięci
34 ...
```

W przykładzie tym wykorzystano obiekt `vtkPoints`, służący do określania geometrii regularnej siatki, o wymiarach zadeklarowanych w postaci tablicy trzech liczb całkowitych (tablica `dims`). W wierszu 5 zadeklarowano wyspecjalizowaną wersję obiektu `vtkDataArray`, który służy do przechowywania wartości typu *float.* Wiersz 6 zawiera wywołanie metody określającej rozmiar krotki, potrzebny do zapisu wartości wektorowej (patrz podrozdz. 3.2.8). Kolejny wiersz (7) zawiera wywołanie metody, która rezerwuje ciągły obszar pamięci o rozmiarach równych rozmiarowi zmiennej `float`, pomnożonej przez liczbę punktów oraz liczbę komponentów w krotce.

W pętli (wiersz 12) wykonywane są obliczenia, kolejno dla wszystkich punktów. Wyliczone wartości współrzędnych punktu oraz składowe wektora wpisywane są do obiektów `points` oraz `vectors` pod ten sam indeks (wiersz 25 i 26), dzięki temu współrzędne oraz atrybuty są ze sobą łączone. Wiersz 30 zawiera kod, który przypisuje współrzędne punktów do obiektu `sgrid`, po czym w następnym wierszu następuje „zezwolenie" na zwolnienie zasobów zajmowanych przez obiekt `points`.

Dane wektorowe dotyczą konkretnych punktów, dlatego w wierszu 32 obiekt `sgrid` najpierw wywołuje metodę „wystawiającą punkty" (`GetPointData()`), po czym następuje wywołanie metody `SetVectors()`, która przypisuje do punktów atrybuty wektorowe. Takie „piętrowe" wywoływanie metod jest bardzo często spotykane w VTK.

**Polygonal Data**

Przykład tworzenia sześcianu opisanego w postaci danych Polygonal Data, przedstawiono w listingu 3.5.

**Listing 3.5:** Przykład tworzenia sześcianu – dane typu Polygonal Data.

```
1  vtkPolyData *cub = vtkPolyData::New();          \\ obiekt opisujący "dane graficzne"
2  vtkPoints *points = vtkPoints::New();           \\ współrzędne wierzchołków (geometria)
3  vtkCellArray *polys = vtkCellArray::New();      \\ sposób powiązania wierzchołków (topologia)
4  vtkFloatArray *scalars = vtkFloatArray::New();  \\ wartości przypisane wierzchołkom
5
6  for (i=1; i<8; i++)
7      points->InsertNextPoint(x[i],y[i],z[i]);    \\ dodanie punktu do listy wierzchołków
8  for (i=1; i<6; i++)
9      polys->InsertNextCell(4,pts[i]);            \\ wyznaczenie komórek z 4 punktów
10 for (i=1; i<8; i++)
11     scalars->InsertTule1(i,i);                  \\ dodanie wartości
12
13 cube->SetPoints(points);                        \\ przypisanie punktów do obiektu graficznego
14    points->Delete();
15 cube->SetPolys(polys);                          \\ przypisanie topologii
16    polys->Delete();
17 cube->GetPointData()->SetScalars(scalars);      \\ dodanie wartości do wierzchołków
18    scalars->Delete();
19 ...
```

Dane typu *Polygonal Data* są przystosowane do szybkiego wyświetlania przez kartę graficzną. Ponieważ najwydajniejszą metodą wyświetlania powierzchni jest obecnie siatka trójkątów, często dane typu Polygonal Data nazywa się *siatkami trójkątów* lub po prostu *siatkami*. Siatki takie są zazwyczaj wynikiem działania algorytmów, które generują powierzchnię na podstawie danych (np. izopowierzchnia), opisu matematycznego (np. elipsoida) lub są wynikiem modelowania przestrzennych brył (np. model przekładni zębatej).

Tworzenie danych typu Poligonal Data można podzielić na następujące etapy:

1. Zdeniowanie geometrii (współrzędnych punktów) poprzez wykorzystanie obiektu `vtkPoints` (wiersz 2), przypisanie punktom współrzędnych (wiersz 8), a następnie przypisanie punktów do danych metodą `SetPoints()` (wiersz 13).

2. Zdeniowanie topologi dla: wierzchołków (ang. *vertex*), linii (ang. *lines*), wielokątów (ang. *polys*) (wiersz 9) oraz pasków trójkątów (ang. *triangle strip*), przy pomocy obiektu `vtkCellArray` (wiersz 3) oraz metod: `SetVerts()`, `SetLines()`, `SetPolys()` (wiersz 15), `SetStrips()`.
3. Stworzenie i przypisanie atrybutów dla punktów (vtkPointData) oraz/lub komórek (vtkCellData). Dostęp do tych danych odbywa się poprzez wskaźniki do obiekty, uzyskiwane metodami `GetPointData()` oraz `GetCellData()`. Na otrzymanych w ten sposób obiektach można wykonać metody: `SetScalars()`, `SetVectors()`, `SetNormals()`, `SetTensors()` oraz `SetTCoords()`. Szczegółowe informacje można znaleźć w dokumentacji klasy `vtkDataSetAttributes`.

**Źródła obiektów graficznych**

Bardzo liczną grupę obiektów w bibliotece VTK tworzą obiekty, które służą do generowania danych na podstawie konkretnych modeli parametrycznych. Są to obiekty, które generują dane obrazowe (ang. *image data*), np. obraz zawierający parametryczną siatkę (`vtkImageGridSource`), rozkład Gaussa (`vtkImageGaussianSource`), szum (`vtkImageNoiseSource`) czy też sinusoidalnie zmienne wartości (`vtkImageSinusoidSource`). Dostępne są również obiekty generujące dane graficzne (ang. *polygonal data*) opisujące podstawowe bryły geometryczne (`vtkLineSource`, `vtkDiscSource`, `vtkCubeSource`, `vtkCylinderSource`, `vtkSphereSource`) oraz bardziej skomplikowane, takie jak: strzałki (`vtkArrowSource`), reprezentacje wersorów układu współrzędnych (`vtkAxes`), powierzchnie parametryczne (`vtkSuperquadricSource`), tekst (`vtkVectorText`) czy nawet globus (`vtkEarthSource`). Przykład wykorzystania parametrycznego źródła danych graficznych znajduje się w listingu 3.1. Dostępna dokumentacja Doxygen oraz strona z przykładami [10] zawierają programy wykorzystujące wszystkie z wymienionych klas.

## 3.4. Obiekty wejścia-wyjścia

Biblioteka VTK posiada obiekty, które umożliwiają odczyt i zapis do plików różnych rodzajów danych. Obiekty te można podzielić na dwie grupy: zapisująco-odczytujące oraz eksportująco-importujące.

### 3.4.1. Odczyt-zapis danych

Obiekty odczytujące i zapisujące podstawowe typy zbiorów danych występują w największej liczbie. Praktycznie każdy typ danych ma wyspecjalizowane obiekty umożliwiające obsługę plików. Obiekty te mają w nazwie słowa *Reader* lub *Writer:*

- dane typu Polygonal Data, zapisujące w formacie VTK, np.: `vtkPolyDataReader`, `vtkPolyDataWriter`, `vtkXMLPolyDataReader`, `vtkXMLPolyDataWriter`, jak również w innych formatach np.: `vtkSTLReader`, `vtkSTLWriter`, `vtkOBJReader`, `vtkPDBReader`, `vtkIVWriter` i wiele innych,

[10] `http://www.vtk.org/Wiki/VTK/Examples`

- dane typu Image Data, operujące standardem VTK np.: `vtkStructuredPointsReader`, `vtkStructuredPointsWriter`, `vtkXMLStructuredPointsReader`, `vtkXMLStructuredPointsWriter` oraz innymi standardowymi formatami np.: `vtkBMPReader`, `vtkBMPWriter`, `vtkPNGReader`, `vtkPNGWriter`, `vtkTIFFReader`, `vtkTIFFWriter`, `vtkJPEGReader`, `vtkJPEGWriter`, `vtkDICOMImageReader`, `vtkVolume16Reader`,
- dane typu Structured i Unstructured Grid, np.: `vtkStructuredGridReader`, `vtkStructuredGridWriter`, `vtkXMLStructuredGridReader`, `vtkXMLStructuredGridWriter`, `vtkUnstructuredGridReader`, `vtkUnstructuredGridWriter`, `vtkXMLUnstructuredGridReader`, `vtkXMLUnstructuredGridWriter` oraz wyspecjalizowane: `vtkPOPReader`, `vtkAVSucdReader`.

Listing 3.6 zawiera fragment programu, wczytującego dane, w tym przypadku obraz z pliku *bmp,* przy użyciu obiektu z rodziny `vtk*Reader`.

**Listing 3.6:** Przykład odczytu danych obiektem typu vtk*Reader.

```
1  vtkBMPReader *bmpReader = vtkBMPReader::New();  // utworzenie obiektu
2    bmpReader->SetFileName ( "plik.bmp" );        // określenie nazwy pliku
3    bmpReader->Update();                          // odczytanie danych z pliku
4  // przykładowe wykorzystanie wczytanych pikseli
5  vtkImageData* image = bmpReader->GetOutput();
6  ...
```

Obiektu typu `vtk*Writer` używa się, podłączając go do wyjścia filtru (listing 3.7).

**Listing 3.7:** Przykład zapisu danych obiektem typu vtk*Writer.

```
1  vtkPolyDataWriter *writer = vtkPolyDataWriter::New(); // utworzenie obiektu
2      writer->SetInput(filtr->GetOutput());       // pobranie danych z obiektu filtr
3      writer->SetFileName("daneWyjsciowe.vtk");   // określenie nazwy pliku
4      writer->SetFileTypeToBinary();              // tryb zapisu danych
5      writer->Write();                            // zapis danych do pliku
```

## 3.4.2. Import-eksport sceny

Obiekty importujące i eksportujące umożliwiają zapisanie lub odczytanie z plików danych dotyczących wszystkich obiektów sceny (aktorów, świateł oraz kamer):

- obiekty wspierające odczyt scen to: `vtk3DSImporter` (dane w standardzie programu 3D Studio) oraz `vtkVRMLImporter`, odczytujący sceny zapisane w formacie VRML 2.0,
- obiekty eksportujące scenę do popularnych standardów są potomkami obiektu `vtkExporter`. Należą do nich takie obiekty jak: `vtkVRLMExporter`, `vtkRIBExporter`, `vtkGL2PSExporter`, `vtkIVExporter` [11].

[11] Pełną listę obiektów potomnych dla dowolnych obiektów znajduje się w dokumentacji online wygenerowanej przez Doxygen.

Obiekty służące do importu i eksportu scen, podłączane są bezpośrednio do okna wizualizacji, co można zauważyć w listingach 3.8 oraz 3.9.

**Listing 3.8:** Przykład importu sceny obiektem typu vtk*Importer

```
vtk3DSImporter *importer = vtk3DSImporter::New();
    importer->SetFileName("iflamingm.3ds");   // nazwa pliku do odczytu
    importer->ComputeNormalOn();              // włączenie wygładzania powierzchni
    importer->SetRenderWindow(renderWindow);  // przypisanie sceny do okna renderWindow
    importer->Read();                         // odczytanie danych
```

**Listing 3.9:** Przykład eksportu sceny obiektem vtk*Exporter.

```
vtkVRMLExporter *exporter = vtkVRMLExporter::New();
    exporter->SetRenderWindow(renderWindow);  // połączenie sceny z obiektem
    exporter->SetFileName("scena");           // nazwa pliku
    exporter->Write();                        // zapis całej sceny do plik VRML
```

### 3.4.3. Zapis wizualizacji do pliku

Biblioteka udostępnia dwie klasy, których zadaniem jest przechwycenie zawartości okna wizualizacji oraz udostępnienie tych danych w postaci obiektu `vtkImageData`.

Klasa `vtkWindowToImageFilter` zapisuje zawartość okna w rozdzielczości równej rozmiarom pikselowym okna. Listing 3.10 przedstawia przykład użycie tego filtru. Podstawową wadą tego obiektu jest to, że nie można zapisać okna wizualizacji o rozdzielczości większej niż rozdzielczość ekranu.

**Listing 3.10:** Przykład zapisu zawartości okna wizualizacji do pliku

```
vtkWindowToImageFilter *win2img = vtkWindowToImageFilter::New();
   win2img->SetInput(renderWindow);              // podłączenie do okna
vtkJPEGWriter *writer = vtkJPEGWriter::New();    // obiekt do zapisu wyników w pliku
   writer->SetInput(win2img->GetOutput());       // pobranie pikseli obrazu
   writer->SetFileName("wizualizacja");
   writer->Write();
```

Drugi obiekt `vtkRenderLargeImage` umożliwia wygenerowanie obrazu o dowolnie dużych wymiarach. Przykład użycia przedstawiono w listingu 3.11. Parametr `SetMagnification()` odpowiada za zwiększenie rozdzielczości obrazu.

**Listing 3.11:** Zapis wizualizacji o wysokiej rozdzielczości do pliku

```
vtkRenderLargeImage *plakat = vtkRenderLargeImage::New();
   plakat->SetInput(renderWindow);              // podłączenie do okna
   plakat->SetMagnification(4);                 // stopień zwiększenia rozdzielczości
vtkTIFFWriter *writer = vtkJPEGWriter::New();   // obiekt do zapisu wyników w pliku
   writer->SetInput(plakat->GetOutput());       // pobranie pikseli zwiększonego obrazu
   writer->SetFileName("wizualizacjaHiRes");
   writer->Write();
```

Inną metodą wygenerowania okna wizualizacji o wysokie rozdzielczości, jest eksport sceny, a następnie użycie zewnętrznego programu do renderingu. Możliwe jest zapisanie sceny w formacie programów *RenderMan* lub *Povray.*

## 3.5. Mechanizm wymiany komunikatów

Programiści VTK wyposażyli bibliotekę w mechanizm umożliwiający wysyłanie grupowego powiadomienia o zmianie stanu konkretnego obiektu do innych obiektów, które o konkretnej zmianie powinny zostać poinformowane. Mechanizm ten jest implementacją tzw. *wzorca obserwatora.* Każda klasa VTK, będąca potomkiem klasy `vtkObject`, posiada metodę `AddObserver()`, która może posłużyć do „zestawienia połączenia" między danym obiektem (rozgłaszającym) a innym (obserwatorem), który ma być powiadamiany. Obiekt *obserwator* otrzymuje zgłoszenia (ang. *callback*) o wszystkich zdarzeniach (ang. *event*) obiektu rozgłaszającego. Jeżeli zdarzenie „pasuje" do tego, na które czeka obserwator, wówczas następuje reakcja obiektu obserwującego.

Stworzenie własnego obiektu obsługującego mechanizm powiadamiania, który będzie obserwatorem, składa się z trzech kroków[12]:

1. stworzenia funkcji obsługującej zgłoszenie, tzw. funkcji `callback`, o następującym prototypie:
   ```
   void funkcja ( vtkObject *, unsigned long eid, void *clientdata,
   void *calldata )
   ```
2. stworzenie obiektu, który będzie odbiorcą komunikatów:
   ```
   vtkSmartPointer<vtkCallbackCommand> keypressCallback =
   vtkSmartPointer<vtkCallbackCommand>::New();
   ```
   a następnie podłączenie stworzonej funkcji:
   ```
   keypressCallback->SetCallback ( funkcja );
   ```
3. zarejestrowanie obserwatora do obiektu rozgłaszającego, np. w taki sposób:
   ```
   renderWindowInteractor->AddObserver ( vtkCommand::KeyPressEvent,
   keypressCallback );
   ```

Wszystkie filtry zgłaszają zdarzenie `ProgressEvent`, które informuje o postępach w obliczeniach. Na zdarzenie to może reagować inna klasa, która przechwyci wartość liczbową, a następnie np. wyświetli ją na konsoli lub zaktualizuje pasek postępu. Przykład całego programu, w którym obserwowany jest postęp działania filtra znajduje się pod adresem `http://www.vtk.org/Wiki/VTK/Examples/Developers/ProgressReport`.

Lista wszystkich zgłaszanych zdarzeń znajduje się w dokumentacji Doxygen klasy `vtkCommand`.

Grupą obiektów, które szczególnie intensywnie wykorzystują mechanizm obserwatora, są obiekty potomne klasy `vtkInteractorObserver`, w tym `vtkWidget`, `vtkAbstractWidget` oraz `vtkInteractorStyle`. Zapewniają one interakcję zarówno ze sceną (np. sterowanie ruchem kamery), jak i sterowanie obiektami sceny (np. zaznaczanie, przesuwanie).

[12] Na podstawie `http://www.vtk.org/Wiki/VTK/Tutorials/Callbacks`

## 3.6. Tworzenie aplikacji

Biblioteka VTK nie jest „programem" a jedynie zbiorem narzędzi, które wspomagają i ułatwiają tworzenie aplikacji do wizualizacji danych. Naukę VTK najlepiej jest rozpocząć od prostych narzędzi, które umożliwiają uruchamianie dostępnych przykładów oraz szybkie testowanie własnych, krótkich programów. Przytoczone do tej pory przykłady były napisane w języku C++, ponieważ jest to język, w którym VTK powstało. Dodatkowo, powinien on być zrozumiały dla osób, które chcą wykorzystywać biblioteką VTK we własnych aplikacjach. Jednak pisanie w C++ nie jest najwygodniejszym sposobem nauki oraz prezentacji przykładów.

VTK dysponuje interfejsami do prostych języków interpretacyjnych, takich jak *Tcl* oraz *Python.* Co ciekawe, ogromna liczba przykładów dostarczonych wraz z kodem źródłowym oraz na stronie z dokumentacją, zachęca do skorzystania właśnie z tych języków. Uruchomienie przykładów w celu zapoznania się z działaniem konkretnych obiektów VTK nie wymaga głębokiej wiedzy na temat Tcl czy Pythona.

### 3.6.1. Języki skryptowe

Konwersja kodu, zawierającego klasy VTK między językami C++, Java, Python oraz Tcl nie nastręcza większych trudności, ponieważ zarówno klasy, jak i nazwy metod są takie same. Różnice są zazwyczaj w składni. Poniżej, znajduje się ten sam kod zapisany w różnych językach, którego zadaniem jest zmiana koloru aktora:

```
   C++   actor->GetProperty()->SetColor(red,green,blue);
   Tcl   [actor GetProperty()] SetColor $red $green $blue
  Java   actor.GetProperty().SetColor(red,green,blue);
Python   actor.GetProperty().SetColor(red,green,blue)
```

Jak widać, różnice w sposobie wywoływania obiektów VTK są niewielkie, co ułatwia analizę ogromnej liczby dostępnych programów przykładowych. Poniżej znajdują się metody uruchamiania skryptów napisanych w językach Tcl oraz Python, dla dwóch systemów operacyjnych Windows oraz Linux (rodzina Ubuntu).

**Tcl**

Na stronie `http:\www.vtk.org` w dziale *Download*, znajduje się skompilowana wersja VTK, przeznaczona dla systemu Windows. Po jej zainstalowaniu dostępny będzie program `vtk.exe`, który jest interpretatorem języka Tcl skompilowanym wraz z obiektami VTK. Po uruchomieniu programu dostępna będzie konsola oraz okno wizualizacji. Użytkownicy systemu Ubuntu, po zainstalowaniu pakietu `vtk-tcl`, lub po skompilowaniu VTK z kodu źródłowego z włączoną opcją wsparcia języka Tcl, również mają do dyspozycji program `vtk`. Po uruchomieniu udostępnia on konsolę Tcl oraz okno wizualizacji. Komenda `source plik.tcl` wydana w konsoli programu `vtk` umożliwia wykonanie programu `plik.tcl`. Możliwe jest również wywołanie interpretatora razem z nazwą pliku: `vtk plik.tcl`. Tcl spróbuje uruchomić program, informując na konsolę o ewentualnych błędach.

**Python**

Użycie tego języka w systemie Windows wymaga ściągnięcia i zainstalowania darmowego środowiska Python ze strony `http:\www.python.org`. Następnie należy dodać do zmiennej systemowej PYTHONPATH ścieżkę dostępu do modułów biblioteki VTK, zazwyczaj *VTK/Warping/Python* lub *vtk/bin*. Uruchomienie programu w języku Python wykonuje się komendą: python `plik.py`.

W systemie Ubuntu, należy zainstalować pakiet `python-vtk` wraz ze wszystkimi niezbędnymi pakietami dodatkowymi. Uruchomienie programu napisanego w języku Python polega na wydaniu w konsoli systemu operacyjnego komendy: `python plik.py` lub samej `./plik.py`, jeśli plik ma nadane prawo do uruchomienia się jako program. Szczegółowy opis konfiguracji znajduje się pod adresem `http://www.vtk.org/Wiki/VTK/Tutorials/PythonEnvironmentSetup`.

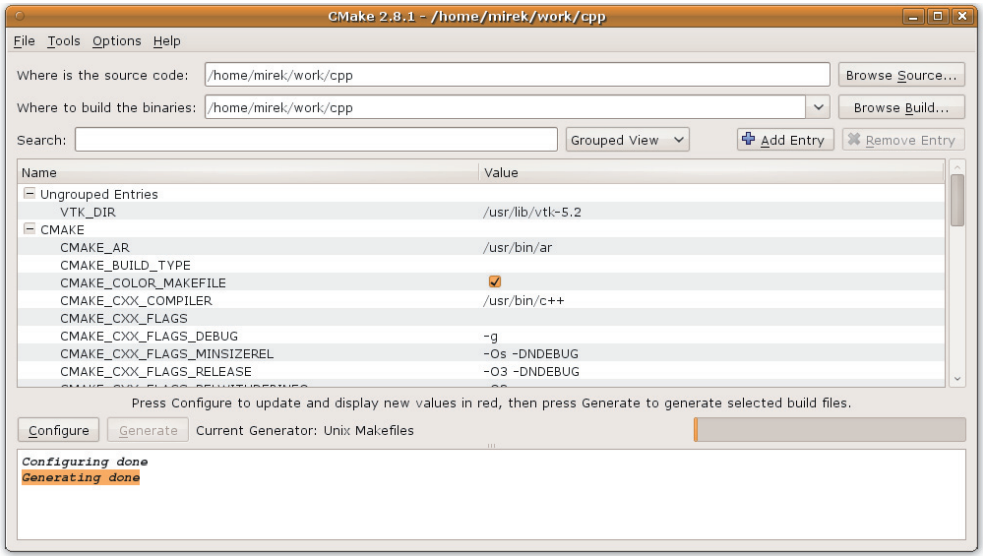
### 3.6.2. C++

Podstawowym narzędziem służącym do przygotowania kodu źródłowego biblioteki VTK do kompilacji, w konkretnym systemie operacyjnym przez wybrany kompilator, jest program *CMake*. Program ten jest dostępny pod adresem `http://www.cmake.org` zarówno w postaci źródeł, jak i w wersji binarnej. Program ten służy do przygotowania kodu źródłowego do kompilacji przez konkretny kompilator.

Do poprawnej konfiguracji kodu źródłowego CMake wymaga, oprócz plików źródłowych, specjalnie przygotowanego pliku projektu o nazwie `CMakeLists.txt`. Przykład zawartości takiego pliku przedstawiono przy okazji prezentacji pierwszego programu VTK (listing 3.1), wraz z niezbędnym plikiem projektu (listing 3.2). Plik projektu jest plikiem tekstowym, zawierającym: najstarszą wersję programu cmake, z którą plik projektu jest kompatybilny, nazwę projektu, deklarację potrzebnych bibliotek (sekcja `FIND_PACKAGE`, zazwyczaj występuje tutaj tylko VTK, jednak może być również ITK, QT lub inna wspierana biblioteka), określenie ścieżki dostępu do plików nagłówkowych (sekcja `INCLUDE`), listę plików składających się na projekt (sekcja ADD_EXECUTABLE) oraz listę bibliotek do zlinkowania (sekcja TARGET_LINK_LIBRARIES, w której zazwyczaj wystarczy dodanie biblioteki vtkHybrid).

Pod adresem `http://www.cmake.org/cmake/help/examples.html` znajduje się opis podstawowych części pliku CMakeLists.txt, zaś pod adresem `http://www.vtk.org/Wiki/VTK/Tutorials/CMakeListsFile` przykład pliku umożliwiającego kompilację programu z obiektami VTK.

Program CMake ma nakładkę graficzną ułatwiającą konfigurację kodu. Po wybraniu katalogu z kodem źródłowym oraz katalogu, do którego zapisana zostanie wersja kodu przygotowana do kompilacji, wybierany jest z listy dostępnych jeden kompilator lub środowisko programistyczne. Program analizuje zawartość plików, a następnie udostępnia wybrane opcje konfiguracyjne (np. wybranie parametrów kompilacji, określenie ścieżek dostępu itp.). Rysunek 3.10 przedstawia ekran programu cmake po konfiguracji oraz wygenerowaniu plików make dla standardowego kompilatora systemu linux'a *Unix Makefile*. Kompilacja programu `sfera` rozpoczyna się po wydaniu polecenia `make`.




**Rys. 3.10.** Przykład konfiguracji kodu źródłowego programem CMake

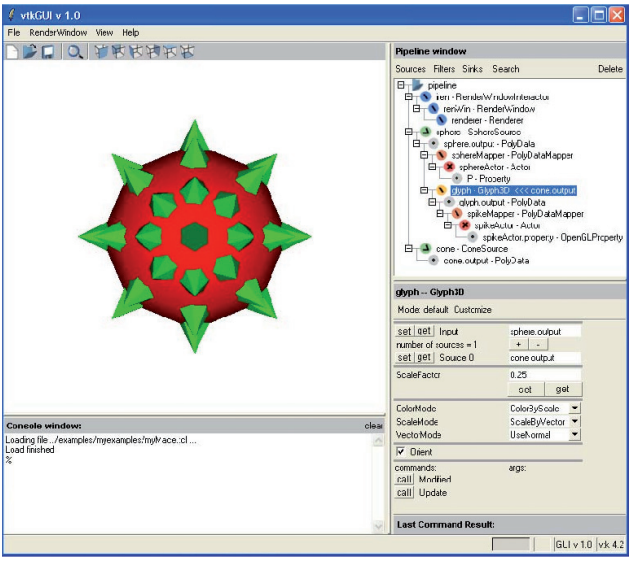
Oczywiście, możliwe jest wykorzystanie dowolnego środowiska programistycznego do pisania programów wykorzystujących obiekty VTK, jednak opis nawet podstawowych środowisk wykracza poza ramy tego opracowania. Osoby zainteresowane zachęca się do poszukania gotowych „projektów szkieletów" w internecie (również na stronie autora).

### 3.6.3. Programowanie graficzne

Ze względu na dość dużą popularność biblioteki Visualization Toolkit, powstało kilka narzędzi graficznych, które umożliwiają *programowanie wizualne* poprzez *rysowanie* strumienia wizualizacji. Do najciekawszych należą:

**vtkGUI** został opracowany w ramach pracy magisterskiej przez *Stefana Svenssona* oraz *Katerinę Vrotsu,* w wyniku współpracy pomiędzy szwedzkim uniwersytetem *Linkoping Tekniika Hogskola* i włoskim konsorcjum *Cineca.* Na stronie `http://vtkgui.cineca.it/download.htm` znajduje się wersja instalacyjna programu vtkGUI oraz rozprawa dyplomowa dotycząca programu. Jest to darmowy program napisany w języku Tcl/IncrTcl, który umożliwia szybkie tworzenie wizualizacji przy użyciu VTK w wersji 4.2.1. Program wprawdzie nie udostępnia rozbudowanych narzędzi do programowania wizualnego, jednak prezentuje strumień wizualizacji w sposób prosty i intuicyjny. Bardzo cenną funkcją programu jest możliwość zapisu strumienia wizualizacji do pliku w postaci kodu C++ lub skryptu Tel. Dzięki temu możliwe jest wykorzystanie vtkGUI do generowania fragmentów kodu większych aplikacji.

Interfejs programu przedstawiono na rysunku 3.11.




**Rys. 3.11.** Interfejs programu vtkGUI

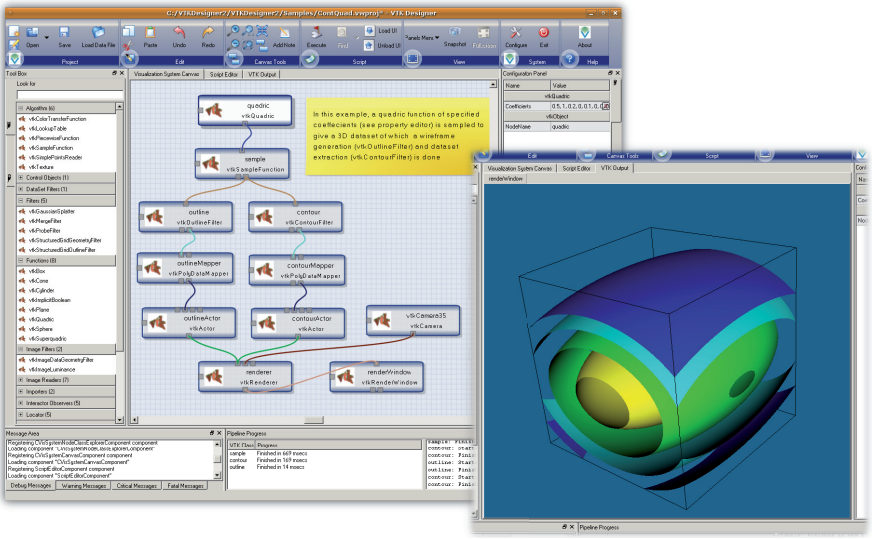
W centrum prezentowane są wyniki wizualizacji strumienia, który budowany i prezentowany jest po prawej stronie interfejsu w postaci drzewa (ang. *pipeline window*). Poniżej znajduje się panel, w którym prezentowane są parametry zaznaczonego w drzewie obiektu. Aplikacja dostarczona jest z dużą ilością przykładów oraz dobrą dokumentacją. Program jest szczególnie polecany do nauki i testowania działania podstawowych algorytmów zaimplementowanych w VTK. Podstawową wadą programu jest brak aktualizacji, wspieranie VTK w starej wersji 4.2 oraz zdarzające się problemy ze stabilnością.

**VTK Designer** jest to darmowy program o dużych możliwościach, który umożliwia programowanie strumienia w sposób wizualny. Interfejs programu przedstawiono na rysunku 3.12. Obiekty strumienia wizualizacji przedstawiane są w postaci „bloczków", które można ze sobą łączyć, tworząc „wizualny program", przetwarzający i wizualizujący dane. Program dostępny jest pod adresem `http://www.vcreatelogic.com/products/vtkd`.

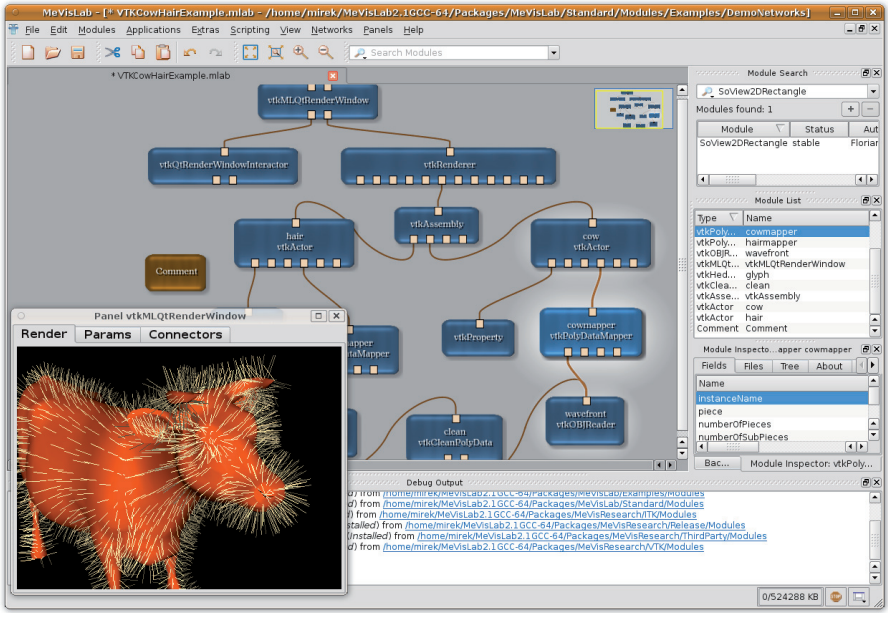
Program ma wiele dobrze opisanych przykładów oraz przystępną dokumentację. Niestety, nie umożliwia eksportu kodu strumienia wizualizacji.

**MeVisLab** to bardzo rozbudowane narzędzie, które umożliwia programowanie wizualne głównie aplikacji medycznych, z wykorzystaniem wielu bibliotek w tym *VTK, ITK, MeVis Image Processing Library* (ML), *Open Inventor* oraz wielu autorskich rozwiązań [13]. Program jest darmowy, przystosowany dla wielu platform (Windows, Mac OS, Linux) i dostępny pod adresem `http://www.mevislab.de`. Interfejs programu przedstawiono na rysunku 3.13.

[13] Szczegóły na stronie `http://www.mevislab.de/mevislab/features/`



**Rys. 3.12.** Interfejs programu VTK Designer




**Rys. 3.13.** Interfejs programu MeVisLab

# Rozdział 4
# Podstawowe metody wizualizacji danych

W rozdziale tym zaprezentowane zostaną podstawy wizualizacji informacji zarówno danych reprezentujących rzeczywistość, jak np. informacje pomiarowe, jak i informacji abstrakcyjnych, np. schemat przepływu informacji.

## 4.1. Zasady tworzenia przekazu graficznego

Francuski kartograf, profesor na Sorbonie Jacques Bertin opublikował w 1967 roku jedno z najważniejszych, zwłaszcza dla graficznej wizualizacji informacji, monumentalne dzieło pod tytułem *Sémiologie Graphique* (ang. *Semiology of Graphics*) [Ber83]. W książce tej, opierając się na własnych doświadczeniach nabytych w zawodzie kartografa, przeanalizował teoretyczne podstawy przekazywania informacji oraz opracował zasady zapisu informacji w postaci dwuwymiarowej grafiki: map, wykresów oraz diagramów.

Semiologia lub inaczej semiotyka (ang. *seminology*) jest teorią badającą wpływ znaków na porozumiewanie się ludzi. Obejmuje ona semantykę, syntaktykę i pragmatykę [DKSSW06]. Opisana przez Bertina seminologia grafiki traktuje wizualizację jako przekaz informacji, kierowanej od autora wizualizacji do jej odbiorcy. Poprawny przekaz informacji jest możliwy wówczas, gdy zarówno autor, jak i odbiorca posługują się tym samym *kodem*, czyli zbiorem zasad zrozumiałym i akceptowanym przez obie strony.

### 4.1.1. Parametry charakteryzujące dane

Bartin przeanalizował różne rodzaje informacji, jakie mogą być wizualizowane w postaci graficznej, a następnie określił zestaw parametrów, które je charakteryzują:

**Niezmienniki** (ang. *invariants*): cechy lub właściwości, które mogą stanowić zmienne niezależne. Są to podstawowe pojęcia informacji graficznej, wspólne dla wszystkich istotnych zależności. Przykładem informacji niezmienniczej jest *czas* na wykresie zmian temperatury w czasie.

**Składowa** inaczej komponent lub cześć (ang. *components*): są to informacje, które można potraktować jako zmienne wzajemnie niezależne oraz zależne. Składowe wyznaczają „stopnie swobody" informacji. Czas i temperatura z wcześniejszego przykładu są komponentami. Jeżeli do informacji o zależności temperatury w czasie zostanie dodana

np. informacja o położeniu geograficznym czujnika (nazwa stacji meteorologicznej), wówczas ta informacja tworzy kolejną składową. Z kolei, uzupełnienie nazwy stacji o jej współrzędne geograficzne (szerokość, długość i wysokość nad poziomem morza) zwiększa liczbę komponentów, powstają trzy kolejne „stopnie swobody". Składowe mogą tworzyć *wymiary danych*, czyli zestawy składowych, które opisują dane w sposób jednoznaczny. Przykładowo, inny zestaw składowych jest niezbędny do przedstawienia informacji o dobowych zmianach temperatury w konkretnej stacji meteorologicznej a inny do uzyskania informacji na temat zmian temperatur dla wszystkich stacji położonych na wysokości ponad 100 m nad poziomem morza.

**Element, kategoria** (ang. *elements*)*:* są to różne, zidentyfikowane części składowych. Konkretne wartości temperatur są elementami składowej *temperatura.* Kategorie stanowią elementy innego typu, na przykład: na wykresie słupkowym dochodów według zawodów, poszczególne zawody (hutnik, rolnik, nauczyciel, itp.) są elementami składowej *praca.*

**Długość składowej danych** (ang. *length*)*:* jest to liczba elementów w komponencie. Przykładowo, na wykresie słupkowym przedstawiającym dochody według zawodów *długość składowej* równa jest liczbie zawodów. Długość składowej danych jest określana jako *liczba wystąpień danych* lub jako *liczba kategorii w składowej.*

**Organizacja komponentów** (składowych danych): jest to *typ składowych* lub po prostu *typ danych*, który można podzielić na następujące kategorie:

- nominalne (ang. *nominal*)*:* dane typu wyliczeniowego (np. górnik, rolnik, emeryt), posiadają dwa najprostsze operatory: = oraz ≠;
- porządkowe (ang. *ordinal*): dane o możliwej do określenia relacji porządku (np. zimno, ciepło, gorąco), dostępne dla nich operatory to: =, ≠ , <, ≤, > oraz ≥;
- przedziałowe (ang. *interval*): dane interwałowe, posiadające skalę z umownie przyjmowanym początkiem, mają operator porównania (=, ≠, <, ≤, >, ≥), podstawowe operatory matematycznie (+, –), ale nie posiadają operatora stosunku; przykładem danych porządkowych są daty; nie można określić stosunku daty 01.04.1998 i 21.06.2006, ale można wyznaczyć ich różnicę, np. określającą liczbę dni między tymi datami;
- ilorazowe (ang. *ratio*): dane mające skalę, które można dodawać, odejmować oraz przede wszystkim, dla których można wyznaczyć stosunek wartości; przykładami takich danych są wzrost, waga; odległość 12 metrów może być opisana jako 10-krotnie mniejsza niż 120 metrów; dane tego typu =, ≠, <, ≤, >, ≥, +, – oraz ÷.

Analiza parametrów charakteryzujących informację ma kluczowe znaczenie dla poprawnej wizualizacji. Niezbędne jest przede wszystkim poprawne określenie składowych danych oraz sposobu organizacji, ponieważ głównie od tych cech zależy sposób wizualizacji.

## 4.1.2. Zmienne wizualne

Zmienne wizualne są to szeroko pojęte *informacje graficzne*, które służą do prezentacji danych. Bertin rozpatruje płaszczyznę, definiowaną przez dwa płaskie wymiary ($x$, $y$), na której *dwa wymiary danych* mogą być prezentowane jednocześnie. Wymiary te mogą być

wykreślone przy użyciu trzech *typów* graficznych elementów podstawowych (tzw. znaków), jakimi są:

- punkt: stanowi miejsce na płaszczyźnie, nie ma teoretycznej długości ani powierzchni;
- linia: posiada długość, którą można zmierzyć na płaszczyźnie, ale nie ma powierzchni;
- obszar: ma mierzalną powierzchnię.

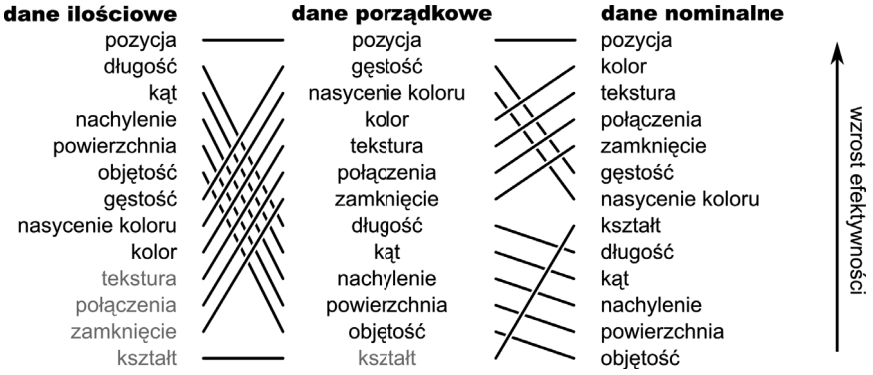
Oprócz przedstawionych powyżej dwóch „płaskich wymiarów", znak graficzny na płaszczyźnie może również przekazywać informacje o kolejnych *wymiarach danych*, za pomocą następujących zmiennych wizualnych (rysunek 4.1):

- rozmiar: wysokość, szerokość, długość, pole powierzchni itp.;
- jasność: różne stopnie, między czarnym i białym;
- kolor: zmienna barwa o jednakowej wartości jasność;
- orientacja: nachylenie lub kąt obrotu linii, kształtów, od pionowej do poziomej;
- struktura: tekstura, cecha powierzchni, która poprzez zmiany w częstotliwości, kierunku lub amplitudzie składników wyznacza obszar o określonej wartości;
- kształt lub forma: koło, kwadrat, gwiazda itp.;
- położenie: pozycja w przestrzeni, wzajemne odległości.


**Rys. 4.1.** Podstawowe zmienne wizualne, tzw. *zmienne siatkówkowe*, na podstawie J. Bertin [Ber83])

Zestaw tych zmiennych Bertin nazwał *zmiennymi siatkówkowymi* (ang. *retinal variable*), ponieważ odpowiadają one podstawowym cechom, jakie można przypisać obiektom postrzeganym przez oko. Opracowana klasyfikacja obejmuje jedynie dwuwymiarową reprezentację obrazu, nie uwzględnia czasu ani możliwości prezentacji danych w przestrzeni trójwymiarowej.

Jock Mackinla [Mac86] rozszerzyły liczbę zmiennych (między innymi o: objętość, kąt miedzy krawędziami, nasycenie koloru) oraz stworzył dla nich ranking użyteczności, z uwzględnieniem trzech podstawowych rodzajów danych. Ranking ten przedstawiony jest na rysunku 4.2.




**Rys. 4.2.** Ranking zmiennych wizualnych dla trzech podstawowych rodzajów danych [Mac86]

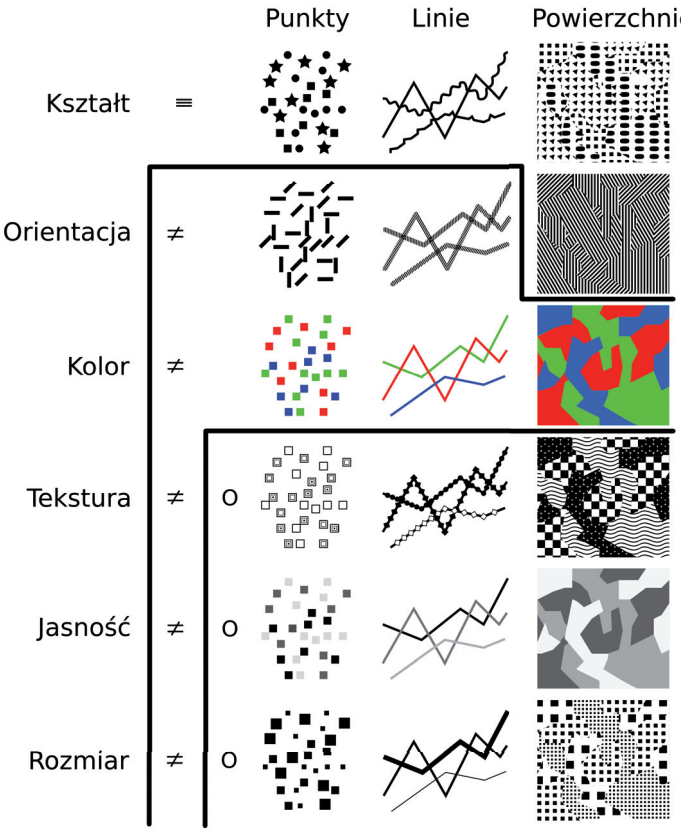
Jak widać, pozycja jest najbardziej uniwersalną zmienną wizualną. Można również zauważyć „zamianę" zmiennych dla danych ilościowych i porządkowych. Kolorem szarym zaznaczono zmienne wizualne, których powinno się unikać przy konkretnym typie danych.

### 4.1.3. Mapowanie informacji

Bertin wyróżnił cztery rodzaje mapowania informacji na wizualną reprezentację z uwzględnieniem czterech rodzajów zależności [Ber83]:

1. Mapowanie **łączne**, asocjacyjne (≡): umożliwiające zobaczenie wszystkich kategorii danych z uwzględnieniem różnic, umożliwia uzyskanie odpowiedzi na pytanie: „Gdzie na mapie jeziora znajdują się łowiska?" lub „Gdzie na mapie jest ziemia?".
2. Mapowanie **selektywne** (≠): prezentujące łącznie wszystkie elementy danej kategorii, umożliwia uzyskanie odpowiedzi na pytania typu: „Gdzie są łowiska łososia?" lub „Gdzie jest pustynia?".
3. Mapowanie **porządkowe** (O – ordered): porządkujące dane w jednym wymiarze np. „Gdzie są większe ryby?" lub „Które regiony lądu są ciepłe, a które gorące?".
4. Mapowanie **ilościowe** (Q – quantitative): ukazujące: stosunek między dwoma elementami wybranego wymiaru, wszystkie elementy lub grupy o podobnych ilościach, np. „Gdzie są wszystkie ryby o długości większej niż jeden metr?" lub „Czy suma opadów na połowie pustyni jest równa opadom w lesie?".

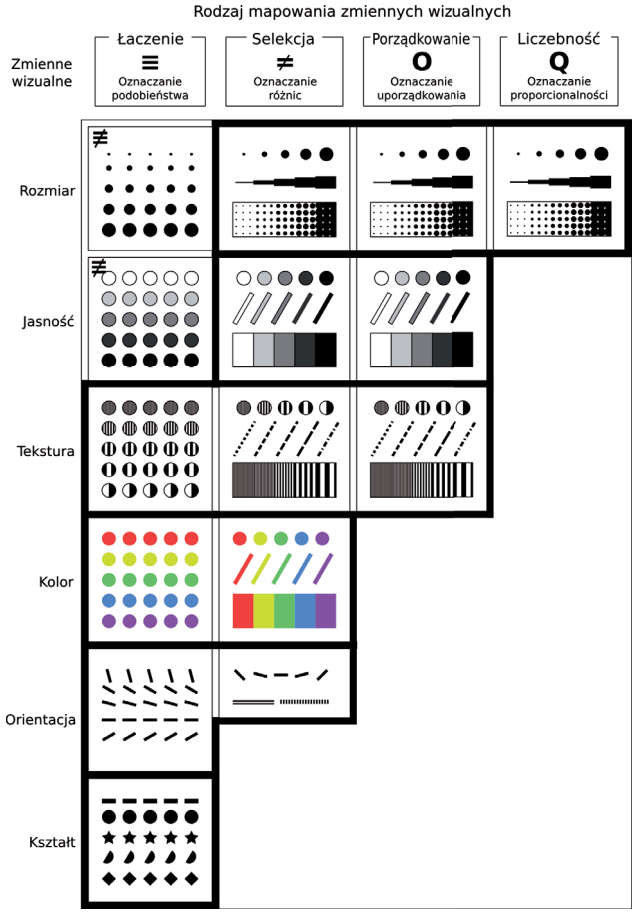
Rysunek 4.3 przedstawia możliwe kombinacje zmiennych wizualnych oraz podstawowych znaków graficznych: punktów, linii oraz powierzchni. Na rysunku zaznaczono również rodzaje mapowań odpowiednie dla poszczególnych kombinacji punkt/zmienna.




**Rys. 4.3.** Tablica Bertina, przedstawiająca możliwe kombinacje znaków zmiennych wizualnych z zaznaczonymi mapowaniami [Ber83]

Analizując zdolność percepcji poszczególnych zmiennych wizualnych, a także uwzględniając mapowania zaproponowane przez Bertina, można sformułować zasady doboru zmienny wizualnych do typu danych. Rysunek 4.4 przedstawia przykłady konstruowania zmiennych wizualnych z użyciem podstawowych znaków (punktach, liniach i powierzchniach) z uwzględnieniem możliwości postrzegania oraz zaznaczonymi metodami mapowania.

Można zauważyć, że *rozmiar* ma najszersze zastosowanie – może służyć zarówno do przedstawiania (mapowania) danych ilościowych (Q), porządkowych (O), jak i mapowania selektywnego. Zmienna ta nie nadaje się do mapowania asocjacyjnego, którego celem jest zobrazowanie wszystkich kategorii danych z uwzględnieniem różnic. Jest to spowodowane trudnościami w percepcji rozmiarów. Również zmienna *jasność* ma takie same ograniczenia – jasność zawsze postrzegana jest w kontekście otoczenia. Również ze względu na ograniczoną rozdzielczość postrzegania kontrastu, zmienna *jasność* nie jest zalecania do prezentacji danych typu ilościowego (Q).

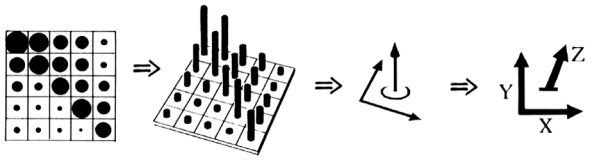



**Rys. 4.4.** Zalecane kombinacje dla podstawowych zmiennych wizualnych i metod mapowania, z uwzględnieniem możliwości postrzegania [Ber83]

### 4.1.4. Skuteczne mapowanie zmiennych

Skuteczność wizualizacji można zdefiniować poprzez *efekty* – jeżeli jedna wizualizacja jest bardziej czytelna niż inna, prezentująca te same dane, wówczas mówimy, że skuteczność pierwszej jest większa.

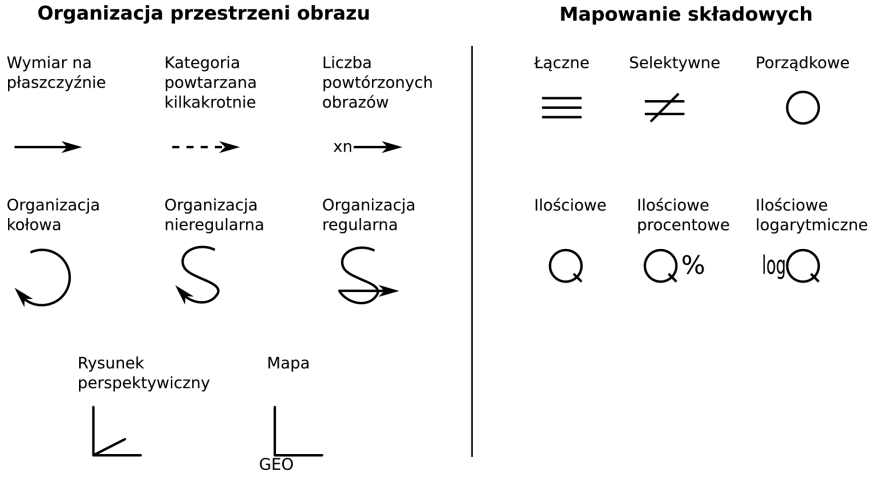
Rysunek 4.5 przedstawia przykład wizualizacji teoretycznych danych trójwymiarowych, w których dwa wymiary zakodowano w postaci pozycji (wiersze i kolumny), zaś do przedstawienia trzeciego wymiaru zastosowano zmienną wizualną *rozmiar*. Jest to wizualizacja zgodna z wytycznymi Bertina. Dane z pierwszego rysunku można przedstawić w postaci wykresu trójwymiarowego. Dwuwymiarowa powierzchnia, na której kodowane są dwa pierwsze wymiary danych, zostaje „perspektywicznie pochylona" (nie zmienia się sposób mapowania), natomiast zmienna wizyjna *rozmiar* zostaje zastąpiona zmienną *długość*.




**Rys. 4.5.** Przykład tworzenia reprezentacji symbolicznej, opisującej organizację elementów graficznych na wykresie [Ber83]

Kolejna część rysunku 4.5 przedstawia symboliczną reprezentację obydwu rodzajów wykresów. Symboliczna reprezentacja wykresów przekazuje jedynie najważniejsze informacje: w jaki sposób zorganizowane jest mapowanie konkretnych wymiarów danych (X,Y,Z). Przedstawiony symbol opisuje więc organizację danych na wykresie.

Na rysunku 4.6 przedstawiono symbole wykorzystywane do oznaczania sposobu organizacji danych na wykresach oraz symbole ułatwiające oznaczenie rodzaju mapowania. Przedstawione symbole tworzą „alfabet", za pomocą którego można opisać wiele rodzajów wizualizacji.

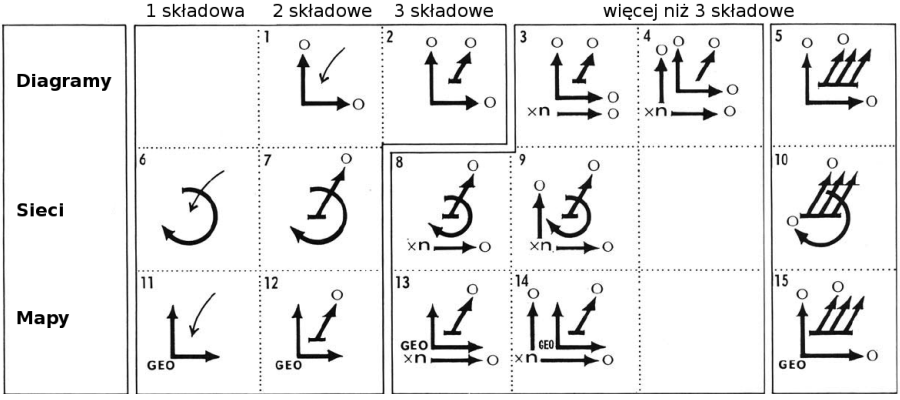



**Rys. 4.6.** Sposób oznaczania organizacji znaków graficznych na płaszczyźnie oraz symbole poszczególnych mapowań [Rob99]

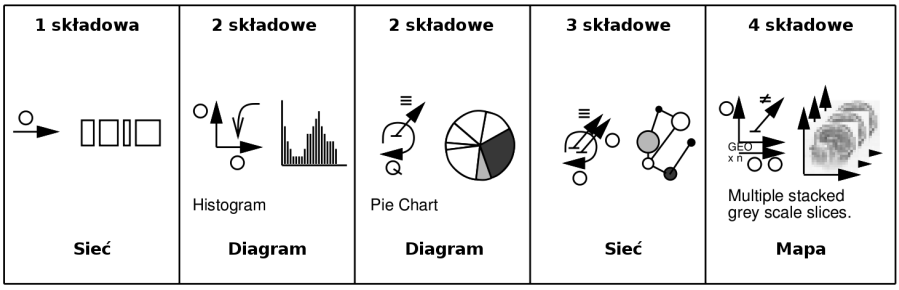
Przykłady wykorzystania powyższych symboli można zobaczyć na rysunku 4.7, który przedstawia podstawowe metody doboru rodzaju prezentacji graficznej (organizacji danych) do *typu danych* oraz *liczby wymiarów.* Rysunek ten przedstawia trzy rodzaje prezentacji graficznych: diagramy, sieci oraz mapy. Dla poszczególnych rodzajów, zaproponowano sposób organizacji, zależny od liczby komponentów (wymiarów) danych. Dodatkowo, wyróżniono typy organizacji danych, które są proste zarówno w tworzeniu, jak i interpretacji (schematy 1, 2, 6, 7, 11 i 12). Większa liczba wymiarów pociąga za sobą konieczność zastosowania

schematów bardziej skomplikowanych, mogących wymagać specjalnych narzędzi do ich tworzenia oraz objaśnienia sposobu „czytania" takiej wizualizacji.

Sposób symbolicznego opisu kilku podstawowych rodzajów wykresów przedstawiono natomiast na rysunku 4.8, zaczerpniętym z pracy [Rob99].




**Rys. 4.7.** Podstawowe metody doboru typu wykresu do rodzaju danych i liczby wymiarów danych [Rob99]




**Rys. 4.8.** Przykłady zastosowania symboli do opisu podstawowych typów wykresów

## 4.1.5. Metody wizualizacji

W pracy [Rob99] przedstawiono zestawienie metod wizualizacji dla podstawowych typów *wizualnej prezentacji danych*. Zestawienie to przedstawia rysunek 4.9. Tabela uwzględnia zaproponowany przez Bertina podział na cztery podstawowe typy wizualnej prezentacji danych:

1. *diagramy*, czyli uniwersalny typ prezentacji graficznej, zazwyczaj dwuwymiarowej, obejmujący: linie czasowe, wykresy (słupkowe, punktowe itp.) czy histogramy; stosowany jest głównie do wizualizacji informacji abstrakcyjnych;

2. *sieci*, będące rodzajem diagramów, które prezentują połączenia (np. listy, drzewa, grafy) między wizualizowanymi danymi, również są stosowane do wizualizacji danych i pojęć abstrakcyjnych;
3. *mapy* to „wyspecjalizowane diagramy" prezentujące dane posiadające *rzeczywiste* współrzędne geometryczne, czyli takie, które powinny być zaprezentowane w sposób bezpośredni; przykładem mogą być fotografie cyfrowe – aby były poprawnie przedstawione, muszą być wyświetlone z zachowaniem proporcji oraz kolejności; geometria danych ma w tym przypadku kluczowe znaczenie;
4. *symbole*, będące zazwyczaj prostymi znakami graficznymi (ang. *glyph*), piktogramami (ang. *icon*);
5. *prezentacje*, łączące powyższe typy podstawowe (np. mapa z zaznaczonym grafem połączeń).

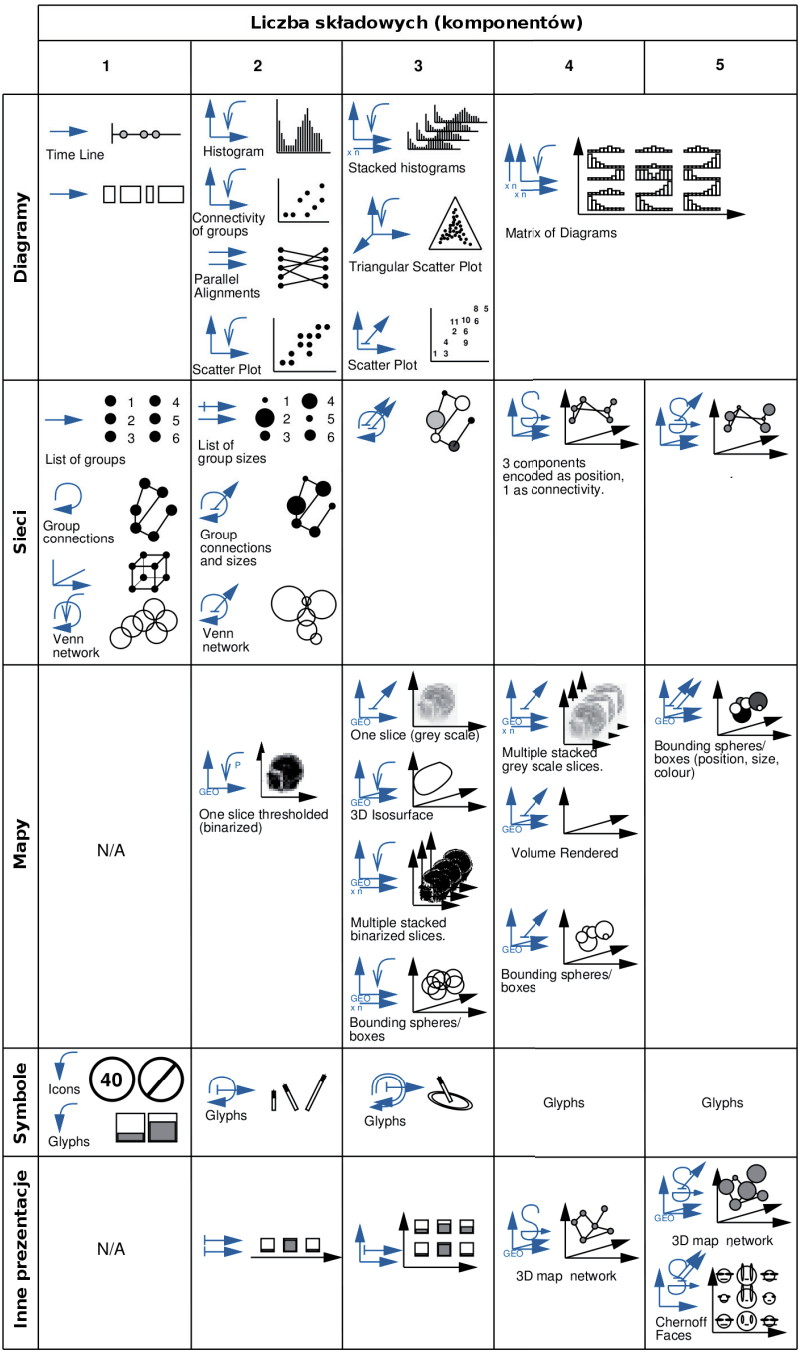
Na rysunku 4.9 kolorem niebieskim zaznaczono symbole określające sposób uporządkowania danych. Obok każdego symbolu znajduje się miniaturowy przykład wizualizacji, spełniającej konkretne uporządkowanie. Przedstawione zastawienie może służyć, jako „katalog" technik wizualizacji, który systematyzuje metody ze względu na rozmiar danych oraz spodziewany rezultat wizualizacji.

W kolejnych częściach skryptu omówione zostaną techniki wizualizacji oraz narzędzia programistyczne umożliwiające uzyskanie większości z zaprezentowanych na rysunku 4.9 typów prezentacji danych.

## 4.2. Możliwości graficzne programu MATLAB

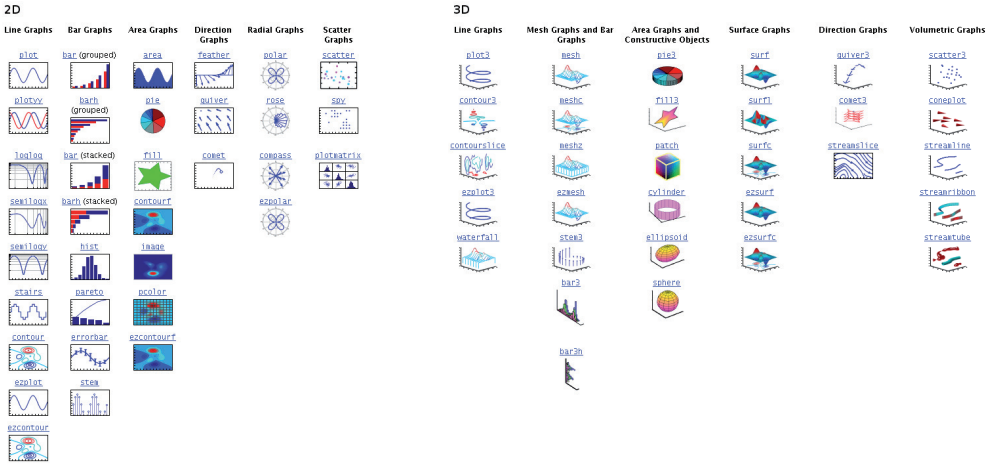
Program Matlab[1] jest powszechnie stosowanym narzędziem w środowisku akademickim. Jest to spowodowane prostym językiem programowania, umożliwiającym wydajne przetwarzanie danych macierzowych i uzupełnionym ogromną ilością specjalistycznych bibliotek, tzw. *toolboxów.* Program integruje biblioteki metod numerycznych, obliczeń symbolicznych, algorytmy optymalizacji, metody rozwiązywania różniczkowych równań cząstkowych, biblioteki statystycznej analizy danych, sztucznych sieci neuronowych, logiki rozmytej, projektowania filtrów, przetwarzania sygnałów, akwizycji obrazów i wiele, wiele innych. Lista dostępnych bibliotek zwiększa się w kolejnych wersjach programu, wprowadzane są między innymi specjalizowane narzędzia do analizy danych biologicznych (ang. *Bioinformatics Toolbox*), przetwarzania obrazów (ang. *Image Processing Toolbox*) czy też analizy danych finansowych (ang. *Financial Toolbox*) i ekonomicznych (ang. *Econometrics Toolbox*). Poza licznymi bibliotekami oraz prostym językiem programowania, środowisko Matlab zawiera moduł *Simulink,* który umożliwia graficzne tworzenie programów, z wykorzystaniem dostępnych toolboxów. Oferowany jest również zestaw narzędzi programistycznych, które umożliwiają kompilację programów napisanych w środowisku Matlaba do niezależnych aplikacji. Możliwe jak również „podłączenie" zewnętrznego kodu do Matlaba i użycie go w postaci kolejnej biblioteki.

[1] Strona domowa programu: `http://www.mathworks.com/`




**Rys. 4.9.** Zestawienie metod wizualizacji dla podstawowych typów prezentacji graficznej [Rob99]

Wśród tych bibliotek znajduje się bardzo obszerny zestaw funkcji odpowiedzialnych za wizualizację wyników obliczeń. Na rysunku 4.10 przedstawiono zestawienie dostępnych w programie *Matlab R2008a* funkcji do wizualizacji danych. Zestawienie to powstało na podstawie dokumentacji dostępnej z programem.




**Rys. 4.10.** Zestawienie funkcji do wizualizacji danych programu Matlab R2008a (na podstawie dokumentacji programu)

Dostępne w Matlabie funkcje można podzielić na dwie grupy: funkcje generujące wizualizacje dwuwymiarowe (moduł *Graphic*) oraz trójwymiarowe (moduł *3-D Visualization*). Opis możliwości poszczególnych funkcji wraz z przykładami zastosowania znajduje się w obszernej dokumentacji programu oraz na stronach internetowych producenta[2].

Poszczególne funkcje zostaną omówione w dalszej części skryptu, przy okazji omawiania metod algorytmów wizualizacji danych.

## 4.2.1. Proceduralna zmiana wyników wizualizacji

Dostępne w programie Matlab funkcje graficzne są bardzo łatwe w użyciu, szybko uzyskiwane są „pierwsze wyniki". Jednak często zdarza się, że uzyskiwane wizualizacje wymagają dopracowania, np. zmiany koloru lub grubości linii. Można te modyfikacje robić ręcznie, z wykorzystaniem dostępnego interfejsu graficznego, jednak przy większej liczbie zmian, działania takie mogą być „stresujące".

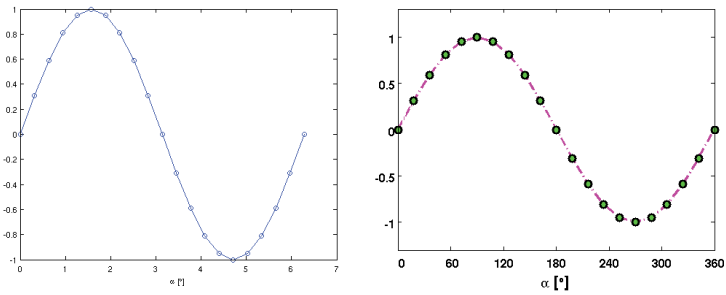
Wszelkie zmiany parametrów obiektów można dokonać z poziomu kodu programu, już w momencie tworzenia skryptu do wizualizacji. Służą do tego funkcje `set()` oraz `get()`. Listing 4.1 przedstawia przykład użycia tych funkcji.

[2] `http://www.mathworks.com/access/helpdesk/help/techdoc/matlab_product_page.html`

**Listing 4.1:** Zastosowanie funkcji `set() get()` programu Matlab do zmiany parametrów wizualizacji

```
clear all;
close all;

%utworzenie przykładowych danych
x = 0:pi/10:2*pi;
y = sin(x);

% wizualizacja danych
figure(1)
h=plot(x,y,'o-'); % uchwyt do wizualizowanej serii danych

% ustalenie parametrów linii i znaczników
set(h,'LineWidth',3, 'LineStyle','-.','Color',[0.9,0.1,0.7])
set(h,'MarkerSize',10,'MarkerEdgeColor','black','MarkerFaceColor','green')

yl = get(gca,'Ylim');                          % pobranie zakresu osi OY
yl_nowy = yl+[-0.3 0.3];                       % nowy zakres osi OY

set(gca,'XLim',[0,2*pi],'Ylim', yl_nowy);      % ustalenie zakresu osi OX i OY
set(gca,'XTick', 0:pi/3:2*pi);                 % zmiana podziałki osi OX

xtl_nowy = 0:60:360;                           %
set(gca,'XTickLabel',num2str(xtl_nowy))      % nowe etykiety osi OY (zamiana
% kąta w radianach na stopnie)

fontSize=get(gca,'FontSize');                  % pobranie wielkości czcionki dla osi
set(gca,'FontSize',fontSize+4, 'FontWeight','Bold') % ustawienie nowych parametrów czcionki osi

xlab = xlabel('\alpha [\circ]');               % uchwyt do etykiety osi OX
set(xlab,'FontSize',18,'FontWeight','Bold')    % ustalenie czcionki dla etykiety osi OX

%% wizualizacja z parametrami domyślnymi
figure(1)
plot(x,y,'o-')
xlabel('\alpha [\circ]');
```

Rysunek 4.11 przedstawia wyniki działania skryptu 4.1.




**Rys. 4.11.** Wynik działania skryptu 4.1

Przedstawiony skrypt, generuje dane zawierające jeden okres sinusoidy (linie 5 i 6). Następnie przedstawia dane w postaci wykresu, używając funkcji `plot()`, jednocześnie pobierając tzw. uchwyt do wykresu. W liniach 13 i 14 funkcja `set()` zmienia parametry linii oraz markerów punktów, używając zapisanego uchwytu. Funkcja `get()` umożliwia pobranie wartości (np. zakresów) z istniejącego już obiektu graficznego. Bardziej szczegółowy opis możliwości zmiany wyników wizualizacji można znaleźć w dokumentacji programu Matlab.

## 4.3. Podstawowe algorytmy biblioteki VTK

W rozdziale 3 omówiono podstawy budowy oraz działania biblioteki VTK. Zaprezentowano tam architekturę strumienia wizualizacji danych, w którym *filtry* stanowią podstawę przetwarzania danych. W kolejnych rozdziałach zostaną omówione poszczególne obiekty biblioteki będące implementacją różnych filtrów, które umożliwiają zastosowanie prezentowanych algorytmów przetwarzania i wizualizacji danych.

## 4.4. Algorytmy wizualizacji danych skalarnych

Jednym z podstawowych typów danych, z jakim mamy do czynienia podczas wizualizacji, są dane skalarne. Są to wszystkie typy informacje, które można zapisać przy pomocy jednej wartości. Wartości te mogą być przypisane zarówno do punktów, jak i komórek (patrz podrozdz. 3.2.5). Ponieważ ten typ danych jest bardzo często stosowany, istnieje wiele algorytmów do ich wizualizacji [SML04].

### 4.4.1. Mapowanie kolorów

Jest to podstawowa metoda prezentacji danych skalarnych. Polega ona na wyznaczeniu dla konkretnej wartości skalarnej, odpowiadającej jej wartości składowych koloru. Mapowanie tego typu implementuje się z użyciem tzw. *metody lookup table.* Na podstawie wartości skalarnej *s* wyznaczany jest indeks *n*-elementowej tablicy. Tablica ta zawiera *n* składowych kolorów (np. RGB), tzw. *paletę kolorów.* Indeks zerowy odpowiada wartość minimalnej *(min*) skalara, zaś ostatnia komórka, o adresie *n* – 1, odpowiada wartości maksymalnej (*max*) skalara. Adres konkretnej komórki *i*, odpowiadającej wartości skalarnej $s_j$ można zapisać w następujący sposób:

$$\begin{cases} i = 0, & \text{if } s_i \leq min \\ i = n-1, & \text{if } s_i \geq max \\ i = n\left(\frac{s_i - min}{max - min}\right), & \text{if } min < s_i < max \end{cases} \tag{4.1}$$

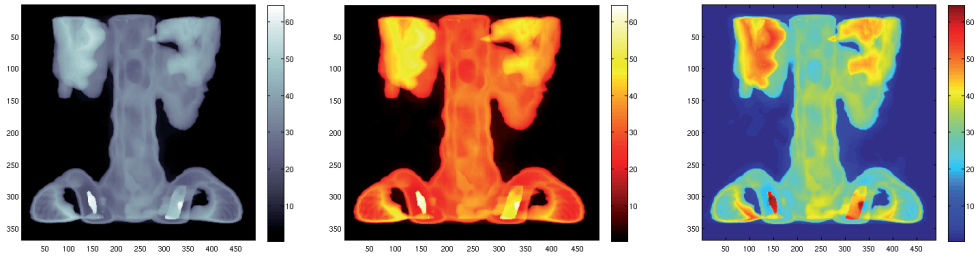
Tak wyznaczony adres komórki *i* tablicy kolorów, zawiera szukane składowe koloru odpowiadające wartości skalarnej. Metoda *lookup-table* jest bardzo wydajna, jednak dla dużej liczby przechowywanych kolorów rozmiary tablicy mogą być znaczne. W najprostszej

wersji algorytmu, przeliczanie wartości skalarnej na indeksy odbywa się z wykorzystaniem zależności liniowej.

Program Matlab zawiera funkcję `colormap`, która jest implementacją liniowej metody mapowania wartości skalarnych na kolory. Przykład użycia tej funkcji zaprezentowano w listingu 4.2. zaś wyniki działania na rysunku 4.12.

**Listing 4.2:** Przykład mapowania kolorów w programie Matlab

```
1  load spine      % wczytanie danych medycznych
2  image(X)        % wizualizacja wczytanych danych w postaci obrazu
3  colormap bone   % zmiana palety barw (mapowania kolorów)
4  colorbar        % włączenie paska, który pokazuje zawartość tablicy kolorów wraz z
     odpowiadającymi wartościami skalarnymi
```



**Rys. 4.12.** Przykład użycia różnych palet kolorów (bone, hot i jet) do mapowania wartości skalarnych

VTK dysponuje kilkoma wersjami algorytmu mapującego kolory, w postaci klas:

- `vtkLookupTable` jest podstawową wersją, która mapuje dane wejściowe do koloru opisanego modelem RGBA: tablica kolorów może być utworzona poprzez wpisanie wartości składowych RGBA lub HSV i wartości przeźroczystości;
- `vtkLogLookupTable` umożliwia mapowanie wartości skalarnych opisanych skalą logarytmiczną;
- `vtkWindowLevelLookupTable` specjalizowana klasa, służąca do określania zakresu wizualizowanych danych nie poprzez wartości minimalną oraz maksymalną, lecz przez zadanie szerokości przedziału oraz jego środka.

Inną metodą mapowania wartości skalarnych do kolorów jest użycie tzw. *funkcji przejścia* (ang. *transfer function*). W metodzie tej, zamiast tablicy z wartościami kolorów, używa się opisu funkcyjnego – każda zwracana składowa koloru może być opisana niezależną funkcją. Daje to znacznie większe możliwości kształtowania wyników mapowania.

Klasa `vtkColorTransferFunction` umożliwia mapowanie wartości skalarnych do kolorów opisanych w przestrzeni RGB lub HSV. Klasa używa wielomianów Hermite'a jako funkcji do interpolacji wartości. Odcinki między zadanymi wartościami (np. metodą `AddRGBPoint( s, r, g, b, midpoint, sharpness)`) mogą przyjmować wartości: stałe, liniowo lub nieliniowo zmienne. Funkcja interpolująca pozwala na określenie punktu środ-

kowego *midpoint* (miejsce, w którym funkcja osiąga średnią z dwóch węzłów ograniczających) jako normalizacji odległość między węzłami oraz wartości współczynnika *sharpness,* odpowiedzialnego za „wygładzenie" odcinka. Uzupełnieniem klasy `vtkColorTransferFunction` o informację o przeźroczystości jest klasa `vtkPiecewiseFunction`, mapująca wartości skalarne do pojedynczej wartości, zwykle wykorzystywanej w kanale przeźroczystości (tzw. *kanał alpha*). Funkcja ta działa podobnie jak poprzednia i również używa wielomianów Hermite'a. Obydwie te funkcje są szczególnie przydatne w renderingu danych objętościowych (wolumetrycznych), gdzie odpowiadają w sposób bezpośredni za wyniki wizualizacji.

Przykłady użycia powyższych klas znajdują się w części poświęconej renderingowi wolumetrycznemu.

Jednym z głównych zagadnień podczas mapowania „skalarów na kolory" jest dobór palety kolorów. Kolor (rozdz. 1) można zapisać w różnych przestrzeniach. Do tworzenia ciągłych przejść między kolorami najlepiej nadaje się opis składowych koloru w przestrzeni HSV. Umożliwia on łatwą kontrolę nad poszczególnymi *zmiennymi wizualnymi* związanymi z barwą: kolorem, nasyceniem koloru oraz jasnością. Przejścia między kolorami należy definiować w taki sposób, by były one proste w interpretacji. Należy również brać pod uwagę aspekty percepcji kolorów, np. fakt istnienia kolorów przeciwstawnych, ciepłych oraz zimnych odcieni barw. Przykładowo, zastosowanie tzw. *palety tęczowej* (zawierającej wszystkie barwy) do prezentacji obrazu termowizyjnego może nie być dobrym rozwiązaniem (zbyt duży „szum" informacyjny). Lepszym rozwiązaniem, będzie użycie skali achromatycznej lub barw od zimnej (niebieskiej) do ciepłej (czerwonej).

### 4.4.2. Konturowanie

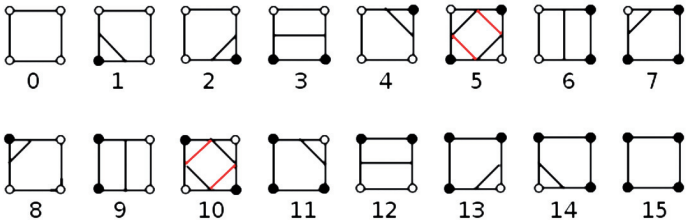
Rozszerzeniem pojęciowym mapowania kolorów są metody *konturowania.* Podczas oglądania „pokolorowanych" wartości skalarnych, zmysł wzroku samoistnie dokonuje grupowania obszarów o podobnych parametrach wizualnych, zgodnie z zasadami teorii percepcji, przedstawionymi w podrozdziale 1.4.1. Takie „grupowanie wzrokowe" prowadzi do wyznaczenia konturów reprezentujących wartości skalarne o podobnych wartościach. Kontury te mogą mieć postać dwuwymiarowych linii oraz trójwymiarowych powierzchni. Przykładem konturów 2D, tzw. *izolinii*, są izotermy rysowane na mapach pogodowych, które łączą punkty o tej samej temperaturze na powierzchni. Rozszerzeniem pojęcia *izolinii* do trzech wymiarów jest *izopowierzchnia.* Przykładem takiego konturu w postaci powierzchni może być np. powierzchnia skóry lub kości wyznaczona na podstawie danych z tomografii komputerowej. Dane tego typu zawierają przestrzenną informację o współczynniku pochłaniania promieniowania X. Ponieważ występuje znaczna różnica między tym współczynnikiem dla powietrza, tkanek oraz kości, możliwe jest określenie takiego poziomu wartości, by izopowierzchnia wyznaczała właśnie powierzchnię skóry. Rysunek 4.16 przedstawia zrekonstruowaną izopowierzchnię, rozpiętą na wartościach odpowiadających granicy tkanki-kości.

Ponieważ wartości skalarne określone są jedynie w skończonej liczbie punktów, wyznaczenie konturu wymaga zastosowania interpolacji. Może być ona obliczona tylko wtedy, gdy znana jest topologia połączeń. Jeżeli dane nie mają informacji o topologii, wówczas niezbędne jest jej wyznaczenie, np. poprzez triangulację. Jak już wcześniej zostało wspomniane,

VTK używa *komórek* do przechowywania informacji o topologii. Dzięki temu, możliwe jest szybkie wyznaczanie wartości pośrednich, np. współrzędnych punktów o zadanej wartości leżących na krawędziach komórki.

Konturowanie można wykonać praktycznie dla dowolnej topologii oraz wymiarze danych. Istnieją jednak dwa szczególnie wydajne algorytmy, które przetwarzają dane obrazowe (ang. *image data*). Są to algorytmy *marching square* dla obrazów 2D o topologii typu *piksel* oraz wersja dla danych 3D (komórek *woksel*) *marching cubes* [LC87].

Podstawowym założeniem tych algorytmów jest fakt istnienia skończonej liczby możliwych przecięć konturu z komórką. Rysunek 4.13 przedstawia wszystkie możliwe sposoby podziału dwuwymiarowej komórki przez kontur.



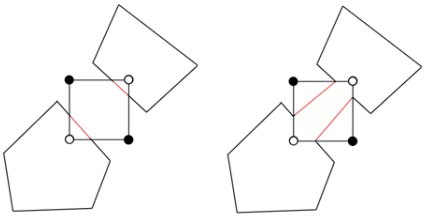

**Rys. 4.13.** Szesnaście możliwych przecięć izolinii z komórką typu piksel [SML04]

Liczba możliwych przypadków zależy od liczby wierzchołków, jakie tworzą komórkę. Każdemu z wierzchołków można przypisać jeden z dwóch stanów: wartość skalarna wierzchołka jest *większa* (●) lub *mniejsza* (○) od zadanej wartości konturu. Jeżeli punkty wyznaczające konkretną krawędź mają różny stan, oznacza to, że na tej krawędzi znajduje się punkt wyznaczający kontur. Współrzędne tego punktu wyznaczane są przez interpolację współrzędnych wierzchołka. Konkretny stan całej komórki można zakodować w postaci liczby binarnej, w której kolejnym bitom przypisane są kolejne stany wierzchołków. Otrzymana liczba stanowi indeks do tablicy opisującej wszystkie możliwe przypadki topologii przecięć konturu z komórką. Podsumowując, algorytm typu *marching* można podzielić na następujące etapy:

1. Wybranie do analizy kolejnej komórki (algorytm „przesuwa się" po kolejnych komórkach, stąd nazwa „kroczący").
2. Wyznaczenie stanu wierzchołków komórki na podstawie porównania wartości skalarnej przypisanej do wierzchołka z zadaną wartością konturu.
3. Stworzenie na podstawie stanu komórek indeksu, określającego stan całej komórki oraz topologię połączeń punktów wyznaczających kontur.
4. Wyznaczenie współrzędnych punktów opisujących kontur.

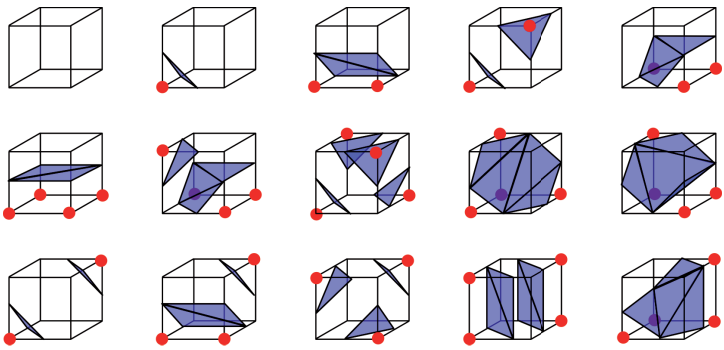
Na rysunku 4.13 przedstawiono 16 możliwych topologii, przy czym można zauważyć, że dwa zaznaczone na czerwono przypadki 5 i 10 są niejednoznaczne. Wyznaczone na krawędziach wierzchołki konturu można połączyć na dwa sposoby. Rozwiązaniem takich niejednoznaczności jest implementacja kilku wersji topologii, które są wybierane, w taki spo-

sób, by np. sąsiednie komórki wyznaczały powierzchnię zamkniętą. Innym podejściem jest analiza wartości skalarów przypisanych do sąsiednich punktów i na ich podstawie wyznaczanie „wagi" (prawdopodobieństwa wystąpienia) dla konkretnej topologii. Rysunek 4.14 przedstawia dwa warianty topologi, których zastosowanie prowadzi do powstania dwóch lub jednego konturu.



**Rys. 4.14.** Dwa warianty topologi połączeń prowadzące do różnych wyników konturowania [SML04

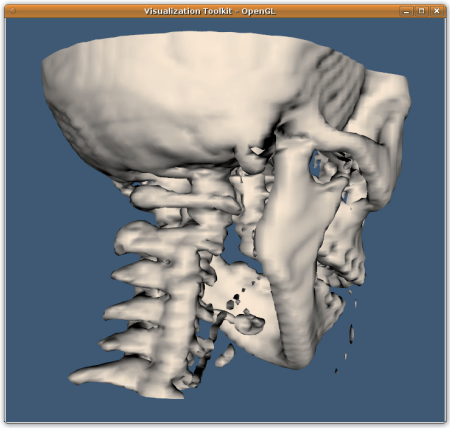
Rozwinięciem metod *marching square* do przestrzeni trójwymiarowej jest algorytm „kroczących kostek" *marching cubes.* Komórka danych obrazowych 3D zawiera osiem wierzchołków, co daje $2^8 = 256$ możliwych przecięć konturu. Liczbę tę można ograniczyć, poprzez uwzględnienie występujących symetrii. Rysunek 4.15 przedstawia 15 podstawowych rodzajów topologii, z których można uzyskać (przez obrót i odbicie) wszystkie 256 kombinacje.



**Rys. 4.15.** Podstawowe przypadki topologii połączeń występujące podczas wyznaczania izopowierzchni metodą *marching cubes*

Rysunek 4.16 przedstawia wynik działania programu, do którego link znajduje się w dokumentacji on-line klasy vtkImageMarchingCubes (`/Graphics/Testing/Tcl/TestImageMarchingCubes.tcl`).

W wyniku działania algorytmu dla poszczególnych komórek powstają niezależne fragmenty opisujące kontur. Implementacja algorytmu jest bardzo wydajna i umożliwia wyznaczanie konturu przez wiele procesów lub nawet maszyn (przetwarzanie rozproszone).




**Rys. 4.16.** Wynik działania programu wyznaczającego izopowierzchnię

VTK zawiera implementację kilku metod wyznaczania konturu:

- `vtkContourFilter` – filtr umożliwia konturowanie danych o dowolnym wymiarze,
- `vtkImageMarchingCubes` klasa specjalizowana, generująca kontur dla danych objętościowych (3D) typu *image data*, ma możliwość strumieniowania danych,
- `vtkMarchingSquares` klasa specjalizowana, do konturowania wyłącznie danych o topologii 2D,
- `vtkSliceCubes` klasa wyspecjalizowana do przetwarzania bardzo dużych zbiorów danych, pobiera jedynie cztery warstwy z danych 3D, na podstawie których wyznacza izopowierzchnię, zapisuje wyniki do pliku, które wczytuje klasa `vtkMCubesReader`.

Klasa `vtkRecursiveDividingCubes` wyznacza kontur poprzez rekursywne dzielenie komórki przecinanej przez kontur. Filtr ten umieszcza punkty w tak podzielonych „podkomórkach". Ciągła izopowierzchnia powstaje przy odpowiednio dużej liczbie generowanych punktów. Wyniki działania tego filtru (zbiór niepołączonych punktów) mogą również służyć jako wejście dla innych algorytmów.

Jeszcze innym podejściem do generowania konturu jest mniej wydajna metoda, opierająca się na „śledzeniu wartości" na krawędziach komórek. Znajdowane na krawędziach punkty, odpowiadające szukanej wartości konturu, łączone są przy użyciu odcinków prostoliniowych lub też dowolną inną krzywą bądź powierzchnią parametryczną.

Program Matlab zawiera następujące funkcje generujące kontury:

- `contour` i `ezcontour` generują dwuwymiarowy kontur przy użyciu linii,
- `contourf` i `ezcontourf` generują kontur 2D używając obszarów,
- `contour3` generuje kontur używając linii oraz dodatkowo umieszczając linie konturów na różnych „wysokościach" wykresu 3D,
- `contourslice` generuje kontur reprezentowany liniami, które leżą na trójwymiarowych płaszczyznach przecinających wizualizowane dane.

## 4.5. Algorytmy wizualizacji danych wektorowych i tensorowych

Dane wektorowe zawierają informację o wartości, kierunku oraz zwrocie i to właśnie te *wymiary* będą prezentowane. Dane wektorowe są również szczególnym przypadkiem danych tensorowych. Ich wizualizacja, szczególnie tensorów wyższych rzędów, stanowi wyzwanie dla metod wizualizacji. Wizualizacje pól wektorowych są coraz częściej spotykane również w medycynie. Głównie w związku z nowymi metodami obrazowania pracy organizmu, takimi jak ultrasonografia Dopplerowska ukazująca np. przepływ krwi, czy też funkcyjny rezonans magnetyczny (fMRI) obrazujący pracę mózgu.

### 4.5.1. Znaczniki – glyph

Podstawową, wpajaną „od przedszkola", reprezentacją wektora jest *strzałka*: zaczepiona w konkretnym miejscu, posiadająca długość oraz zakończona grotem, który wyznacza kierunek wektora. Podobny przekaz niosą różnego rodzaju *znaczniki* graficzne. Do najprostszych zalicza się zorientowaną linię, z zaznaczonym miejscem zaczepienia. Stosowane są również inne kształty, zarówno płaskie (np. trójkąt) jak i przestrzenne (np. ostrosłup). Warunkiem użycia znacznika o konkretnym kształcie, jest możliwość jednoznacznego przekazania podstawowych informacji, jakie niesie ze sobą wektor: miejsca zaczepienia, długości oraz kierunku.

Pakiet Matlab zawiera kilka funkcji, umożliwiających wizualizację danych wektorowych w postaci znaczników:

- `quiver` i `quiver3` umożliwiają prezentację wektorów (np. pola wektorowego) na płaszczyźnie oraz w przestrzeni, poprzez określenie długości współrzędnych wektorów oraz ich punktów zaczepienia, dostępna jest tylko reprezentacja w postaci strzałki;
- `feather` umożliwia przedstawienie ciągu dwuwymiarowych wektorów zaczepionych wzdłuż jednej prostej, równoległej do osi $X$, przykładem takich danych może być wizualizacja kierunku wiatru dla kolejnych dni;
- `compass` pozwala przedstawić grupę wektorów w cylindrycznym układzie współrzędnych, używany jest głównie do wizualizacji kierunku.

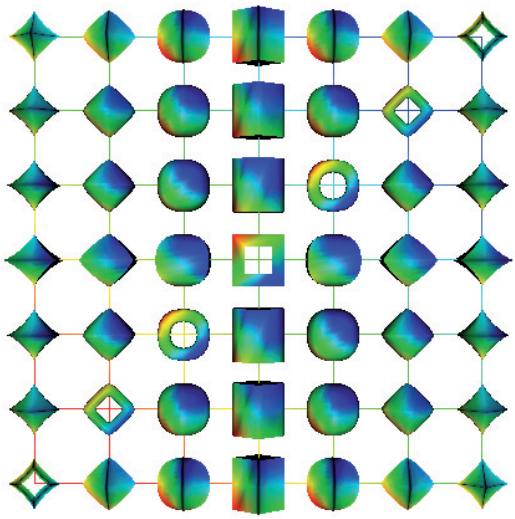
Biblioteka VTK wspiera wizualizację danych wektorowych za pomocą następujących obiektów:

- `vtkGlyph3D` to filtr, umożliwiający wielokrotne skopiowanie dowolnego znacznika (ang. *glyph*) do każdego punktu danych obiektu wejściowego. Przykładem działania takiego filtru może być mapa pogodowa, na której znacznikiem w postaci strzałki zaznaczono prędkość oraz kierunek wiatru zmierzone w konkretnych stacjach meteorologicznych (współrzędne stacji są „punktami" obiektu wejściowego). Znacznik może być dowolnym typem danych *polygonal data*, które są zazwyczaj generowane przez obiekty typu źródła danych (np. `vtkConeSource`, `vtkArrowSource`). Podczas kopiowania znacznika może on zostać zorientowany zgodnie z wektorem normalnym lub wektorem przypisanym do punktu. Znacznik może być również przeskalowany proporcjonalnie

do wartości skalarnej lub długości wektora. Dodatkowo, możliwe jest użycie tablicy znaczników, która adresowana jest wartościami skalarnymi lub długością wektora. Jak widać, możliwości konfiguracji obiektu są duże, dlatego jest on często wykorzystywany do wizualizacji danych o większej liczbie wymiarów.

- `vtkProgrammableGlyphFilter` również umożliwia kopiowanie znaczników do współrzędnych wierzchołków obiektu wejściowego, jednak możliwa jest pełna kontrola tego procesu. Poza obracanie znacznika czy też jego skalowaniem możliwa jest jego dowolna, zaprojektowana przez programistę zmiana. Na rysunku 4.17 przedstawiono wynik działania programu demonstracyjnego `/Graphics/Testing/Tcl/progGlyphsBySource.tcl` dostępnego na stronie on-line dokumentacji VTK. Program ten, na podstawie indeksu danych, modyfikuje parametry znacznika. Znacznikiem w tym programie jest klasa `vtkSuperquadricSource`, będąca źródłem płaszczyzny parametrycznej. Program nie przedstawia „konkretnych" danych, jest jedynie przykładem użycia klasy `vtkProgrammableGlyphFilter`.

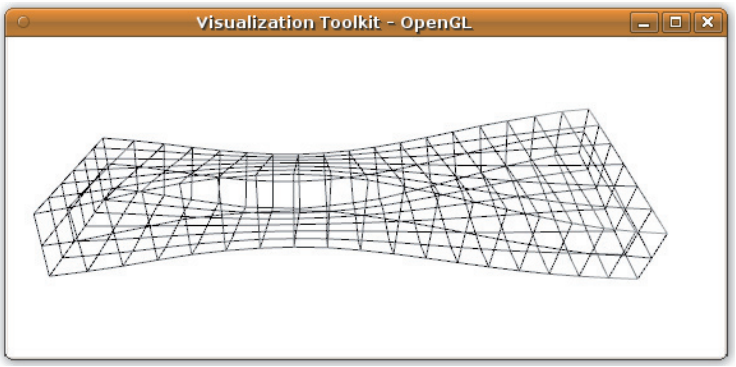
Analizując przedstawiony dwuwymiarowy obraz (rys. 4.17) wygenerowany przy użyciu klasy `vtkProgrammableGlyphFilter`, można się „dopatrzeć" w nim bardzo wielu zmiennych wizualnych – zastosowano mapowanie kolorów, wraz z pozycją znacznika zmienia się jego rodzaj, parametry kształtu oraz jest on obracany. Elastyczność oraz możliwości biblioteki VTK wręcz zachęcają do przedstawiania jak największej liczby *wymiarów danych* na jednej grafice. Często takie podejście umożliwia zobaczenie „drugiego dna" danych. Zawsze jednak należy pamiętać o ograniczonych możliwościach percepcji skomplikowanych wizualizacji. Parametryczne sterowanie znacznikiem jest szczególnie przydatne do przedstawiania wielowymiarowych danych medycznych.



**Rys. 4.17.** Przykład wykorzystania klasy `vtkProgrammableGlyphFilter` do wizualizacji danych wielowymiarowych

### 4.5.2. Wyginanie – warping

Jeżeli dane wektorowe przedstawiają informację na temat ruchu (np. wyniki symulacji drgań, odkształceń, przepływu), wówczas można wykorzystać algorytmy bazujące na modyfikowaniu współrzędnych (topologia oraz atrybuty pozostają bez zmian). Przykładem takiego filtra może być klasa `vtkWarpVector`, która „wygina geometrię obiektu". tzn. wylicza nowe współrzędne wierzchołków przesuwając je w kierunku określonym przez wartość wektorową oraz na odległość proporcjonalną do długości wektora i współczynnika skalującego. W dokumentacji klasy znajduje się link do programu `/Graphics/Testing/Tcl/dispPlot.tcl`. którego wynik działania przedstawiono na rysunku 4.18.




**Rys. 4.18.** Przykład wygięcia siatki przy użyciu klasy `vtkWarpVector`

Istnieje również klasa `vtkWarpScalar`, która przesuwa punkty wzdłuż wektora normalnego, proporcjonalnie do wartości skalarnej. Do nieliniowych parametrycznych deformacji obiektów służą specjalizowane wersje klasy `vtkWarpTransformation`.

### 4.5.3. Linie prądu – streamlines

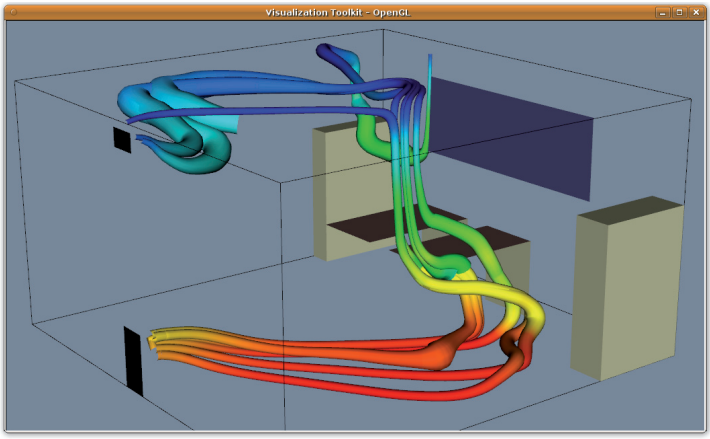
Szczególnie użyteczną techniką prezentacji pola wektorowego są tzw. linie prądów lub linie opływowe (ang. *streamlines*). W metodach tych, wyznaczane są krzywe, reprezentujące teoretyczne trajektorie, po których poruszałyby się cząsteczki o zerowej masie. Podczas wizualizacji wyznacza się pęk krzywych, które zazwyczaj lepiej tłumaczą zjawiska występujące w przepływach. Trajektorie te są wyznaczane poprzez numeryczne całkowanie i dlatego są jedynie aproksymacjami rzeczywistych linii prądów.

Do wyznaczenia linii prądu niezbędne jest określenie w przestrzeni punktu startowego, w którym rozpoczęte zostaną obliczenia. Dodatkowo, określa się kierunek obliczeń (zgodny z przepływem, przeciwny do obydwu) oraz inne parametry (np. warunki początkowe algorytmu całkującego, krok całkowania itp.). W wyniku obliczeń otrzymuje się ciąg punktów w przestrzeni, które opisują trajektorię ruchu cząsteczki oraz przypisane tym punktom inne wielkości, np. prędkość chwilową, rotację itp. Są one wizualizowane poprzez przypisanie

koloru, zastosowanie znaczników lub wygenerowanie wzdłuż trajektorii obiektów geometrycznych, np. „tuby" kodującej prędkość chwilową lub „wstęgi" prezentującej skręcenie.

Biblioteka VTK udostępnia klasę `vtkStreamLine`, która umożliwia wygenerowanie linii prądu dla dowolnego zbioru danych. Klasa `vtkDashedStreamLine` generuje „przerywane" trajektorie, których przerwy znajdują się w jednakowych odstępach czasu. Dzięki temu możliwe jest zaobserwowanie zmian w prędkości na poszczególnych liniach. Kolejną klasą jest `vtkStreamPoints` generująca pojedyncze punkty, reprezentujące konkretne położenie równo oddalone w czasie. Wyjściowy zbiór punktów obliczany przez `vtkStreamLine` można „przyozdobić" filtrami `vtkTubeFilter` oraz `vtkRibbonFilter`. Rysunek 4.19 przedstawia wyniki działania klasy `vtkStreamLine` oraz `vtkTubeFilter` otrzymane po uruchomieniu programu `/VTK/Examples/VisualizationAlgorithms/officeTubes.py`.

W środowisku Matlab dostępne są dwie funkcje `streamslice` oraz `streamline`, generujące linie przepływów. Ich uzupełnieniem są funkcje: `coneplot`, umożliwiająca umieszczanie znaczników na trajektoriach oraz `streamribbon` i `streamtube`, generujące wstęgę oraz tubę wzdłuż linii prądu.




**Rys. 4.19.** Przykład linii prądów wyznaczonych dla numerycznego modelu działania wentylacji w pomieszczeniu biurowym

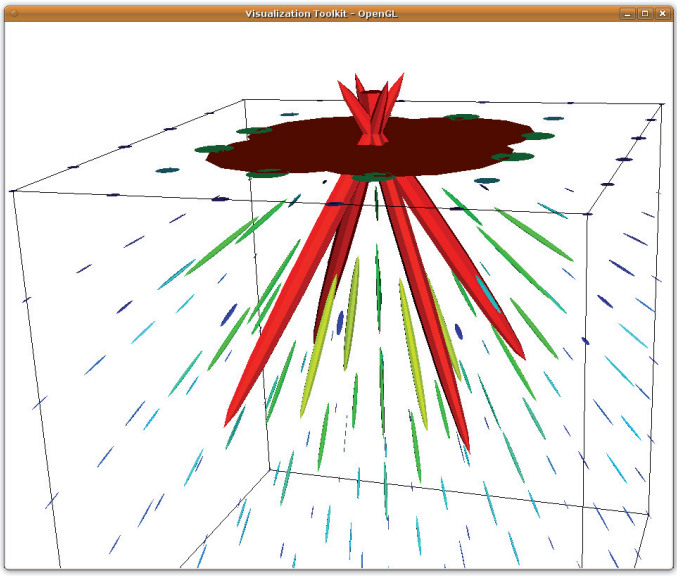
## 4.5.4. Elipsoidy tensorowe

Tensory, szczególnie wyższych rzędów, są sporym wyzwaniem dla algorytmów wizualizacji. Wiąże się to z dużą liczbą wymiarów, które należy przypisać do zmiennych wizualnych. Jednakże istnieje kilka prostych algorytmów, umożliwiających wizualizację symetrycznych tensorów o rozmiarach 3 × 3. Są to tensory opisujące między innymi naprężenia oraz przemieszczenia w trójwymiarowych obiektach. Jedną z tych metod jest metoda elipsoidy.

Symetryczną macierz o wymiarach 3 × 3 można opisać w postaci trzech *wektorów* oraz wartości własnych. Jeżeli potraktujemy wektory własne jako lokalny układ współrzędnych, wówczas wartości własne można potraktować jako współczynniki skalujące w lokalnym

układzie współrzędnych. Podczas wizualizacji, pierwszym krokiem jest, wyznaczenie macierzy oraz wartości własnych. Kształt elipsoidy określany jest, przez wartości własne tensora, zaś za orientację w przestrzeni odpowiadają wektory własne. Elipsoida otrzymywana jest, przez transformację skalującą oraz obracającą sferę.

Rysunek 4.20 przedstawia wizualizację tensorów naprężeń, uzyskaną w wyniku działania programu `/VTK/Graphics/Testing/Tcl/TenEllip.tcl`. Za wyznaczenie elips reprezentujących wartości tensorowe odpowiada klasa `vtkTensorGlyph`.




**Rys. 4.20.** Wizualizacja tensora naprężeń

# Rozdział 5
# Zaawansowane algorytmy wizualizacji

Przedstawione w poprzednich rozdziałach techniki wizualizacji mogą być uznane za metody popularne, szeroko wykorzystywane do różnych celów. Istnieją jednak metody, które znalazły szczególne zastosowanie w medycynie. Są to: *wyświetlanie oświetlonych powierzchni* (ang. *Surface Shading Display – SSD*), częściowo omówiony we wcześniejszym rozdziale przy okazji omawiania konturowania oraz *rendering objętościowy* (ang. *Volume Rendering – VR*), będący głównym tematem tego rozdziału.

## 5.1. Rozwój kart graficznych

Zadaniem pierwszych *kart graficznych* było wyświetlanie tekstu – zamiana kodów znakowych na takie sygnały sterujące monitorem, które na ekranie luminoforu dawały obraz znaków. Liczba dostępnych znaków oraz ich wygląd były z góry ograniczone. Przykładem może być opracowana w 1981 roku przez IBM karta *MDA* (ang. *Monochrome Display Adapter*). Mogło by się wydawać, że takie podejście to „prehistoria", jednak rozwiązania oparte na sprzętowych generatorach znaków są nadal powszechnie stosowane, np. w wyświetlaczach „automatów do kawy"[1]. Kolejnym krokiem w rozwoju sprzętu graficznego było umożliwienie wyświetlania na ekranie zawartości *bufora ekranu (obrazu)*, czyli obszaru pamięci, w którym poszczególnym bitom odpowiadały punkty na ekranie. Dzięki temu, możliwe było wyświetlenie dowolnych znaków, w tym również grafiki. Wypełnianie zawartości bufora ekranu było zadaniem albo sprzętowego generatora znaków, który „zapalał" w buforze obrazu odpowiadające kodom znaków bity, albo zadaniem oprogramowania, które „rysowało" w buforze ekranu: dowolne punkty, linie, okręgi, itp. Z czasem zastosowano w buforze obrazu więcej bitów, umożliwiających przypisanie pojedynczemu pikselowi odcieni szarości. Opracowanie kolorowego kineskopu było kolejnym milowym krokiem. Karta graficzna mogła wygenerować trzy niezależne sygnały, które sterowały kolorem punktu na ekranie. Pierwszym podejściem do generowania sygnału kolorowego było zastosowanie tablicy kolorów – w buforze ramki umieszczany był adres koloru, którego składowe R, G i B odczytywane były z tablicy kolorów. Wartościami z tablicy sterowany

[1] Są to tzw. wyświetlacze alfanumeryczne LCD lub LED ze sprzętowym generatorem znaków, np. układem kompatybilnym ze sterownikiem HD44780

był układ przetwornika cyfrowo-analogowego, sterującego pracą kineskopu. Później, wraz z dostępnością coraz tańszych pamięci, bufor ramki został rozbudowany do znacznych rozmiarów – każdy punkt obrazu mógł być opisany niezależnie, np. trzema ośmiobitowymi wartościami składowych R, G i B. W międzyczasie nastąpił gwałtowny rozwój rynku gier komputerowych oraz graficznych systemów operacyjnych. Programiści opracowali zestawy funkcji np. rysujących podstawowe obiekty graficzne (jak linie), potrzebowali jednak takich rozwiązań sprzętowych, które zwolniłyby zasoby głównego procesora z zadań tak „banalnych", jak rysowanie linii w buforze ekranu. Nastała era akceleratorów 2D (lata 80. ubiegłego wieku), czyli kart graficznych, posiadających specjalizowane układy, które potrafiły np.: „same" narysować linię między dwoma punktami ekranu, wypełnić wielokąt kolorem, wydajnie kopiować fragmenty bufora ramki, czy też sprzętowo obsługiwać rysowanie kursora myszki.

Kolejnym krokiem rozwoju kart graficznych było wprowadzenie akceleracji wyświetlania grafiki trójwymiarowej [2]. Pierwsze rozwiązania opracowane na potrzeby wojska, przemysłu samochodowego oraz lotnictwa (symulatory lotu) były bardzo kosztowne. Jednak zapotrzebowanie rynku konsumenckiego na sprzęt do rozrywki doprowadziło do zaprezentowania w połowie lat 90. ubiegłego wieku tanich kart graficznych (np. S3 ViRGE) ze sprzętową akceleracją grafiki 2D i 3D. Procesor komputera dostarczał jedynie opis trójwymiarowej sceny (współrzędne wierzchołków, tekstury, parametry prostych efektów typu mgła), zaś karty generowały jej obraz przy użyciu stałego i mocno ograniczonego zestawu funkcji.

Współczesne karty graficzne posiadają specjalizowane, wielordzeniowe procesory graficzne (ang. *Graphics Processing Unit – GPU*) o stopniu skomplikowania oraz wydajności znacznie przekraczającej wydajność procesora głównego (ang. *Central Processing Unit – CPU*). Coraz częściej są one wykorzystywane do obliczeń, które nie są bezpośrednio związane z grafiką. Dzięki możliwości niezależnego zaprogramowania wielu procesorów, procesory GPU mogą bardzo wydajnie przetwarzać dowolne dane.

Wraz z rozwojem możliwości tworzenia grafiki komputerowej, powstawały biblioteki standaryzujące dostęp do funkcji sprzętowych. Najstarszą i jednocześnie bardzo uniwersalną jest OpenGL (ang. *Open Graphics Library*). Biblioteka ta została opracowana przez firmę *Silicon Graphics Inc.* na początku lat 90. XX wieku. Znalazła na początku szerokie zastosowanie w profesjonalnych stacjach roboczych, służących do pracy z programami wspomagającymi projektowanie typu CAD (ang. *Computer Aided Design*), do wizualizacji na potrzeby kina i nauki czy też symulatorami lotu. W chwili obecnej implementacja OpenGL dostępna jest na większości nowoczesnych platform sprzętowych, od telefonów komórkowych poprzez komputery klasy PC do superkomuterów, praktycznie niezależnie od systemu operacyjnego. OpenGL jest obecnie rozwijany przez producentów sprzętu graficznego, zrzeszonych w organizacji Khronos [3]. Inną biblioteką oferującą interfejs programistyczny dla aplikacji używających akceleracji kart graficznych jest DirectX firmy Microsoft. Biblioteka ta dostępna jest jedynie w systemach operacyjnych Windows i wykorzystywana jest głównie do tworzenia gier komputerowych.

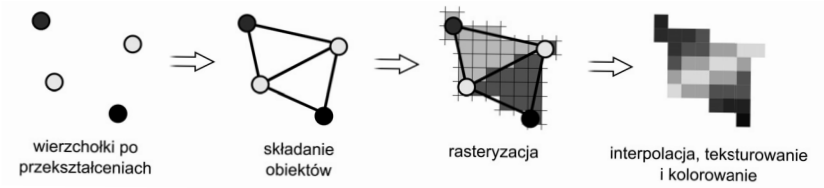
---

[2] Znaczący udział miał w tych pracach Polak Marek Hołyński, `http://pl.wikipedia.org/wiki/Marek_Holynski`

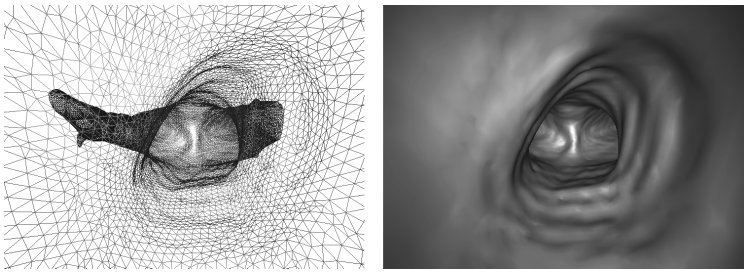
[3] Oficjalna strona organizacji: `http://www.khronos.org`

## 5.2. Rendering powierzchni

Współczesne karty graficzne stanowią wyspecjalizowane koprocesory wspierające praktycznie wszystkie etapy przetwarzanie i tworzenia grafiki trójwymiarowej. Zoptymalizowane są do przetwarzania scen opisanych powierzchniami trójkątów. Podstawowe etapy pracy karty graficznej można sprowadzić do kilu kroków, przedstawionych na rysunku 5.1. Pierwszym jest obliczenie koloru poszczególnych wierzchołków. Zależy on od parametrów wierzchołka, oświetlenia, przyjętego modelu cieniowania. Następnie trójkąty są składane w obiekty wyższego rzędu (linie, trójkąty, wielokąty), te zaś są rasteryzowane. Operacja ta polega na przedstawieniu obiektów opisanych parametrycznie (np. zmiennoprzecinkowymi współrzędnymi końca linii) na urządzeniu rastrowym o skończonej rozdzielczości (jakim jest monitor) – obiekty geometryczne dzielone są na *fragmenty.* Fragmenty są *kandydatami* konkretnego obiektu na, piksele obrazu. Zostaną one wyświetlone, jeśli nie są np. zasłonięte przez inne obiekty. Na koniec, na fragmentach dokonywane są interpolacje kolorów oraz dodawane są tekstury. Nawet kilka wierzchołków może wygenerować miliony fragmentów, jeśli rozdzielczość ekranu będzie znaczna, a obiekty rozpięte na tych wierzchołkach będą odpowiednio duże, np. będą pokrywały cały ekran. Rysunek 5.2 przedstawia wyniki działania karty graficznej na przykładzie powierzchni zrekonstruowanej za pomocą algorytmu konturowania *marching cubes.*



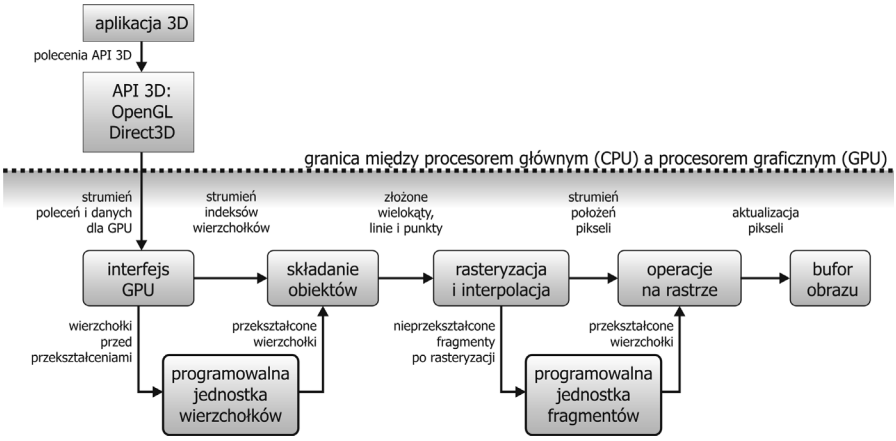

**Rys. 5.1.** Podstawowe etapy pracy karty graficznej [RK03]



**Rys. 5.2.** Wizualizacja, siatki trójkątów przedstawiających izopowierzchnię, w postaci krawędzi oraz trójkątów

Oczywiście, przedstawiony powyżej sposób generowania obrazu trójwymiarowej sceny opisanej wielokątami i wierzchołkami jest bardzo skrótowy i niepełny. Przestawiony na rysunku 5.3 schemat, przedstawia kolejne kroki *klasycznego* przetwarzania danych opisują-

cych scenę 3D. Klasyczność tego algorytmu polega na tym, że częściowo pokrywa się on z wcześniejszymi algorytmami realizowanymi sprzętowo w akceleratorach nieprogramowalnych. Algorytmy te (np. obliczanie cieniowania płaskiego) są obecnie realizowane w postaci programów wykonywanych w jednostkach programowalnych.




**Rys. 5.3.** Programowany potok graficzny realizowany przez kartę graficzną [RK03]

Przedstawiony na rysunku 5.3 schemat posiada dwie rozdzielone grupy jednostek: przetwarzających geometrię sceny (programowalna jednostka wierzchołków) oraz przetwarzających fragmenty obiektów (programowalna jednostka fragmentów). Programy realizujące obliczenia w tych jednostkach nazywane są *programami cieniującymi* wierzchołki (ang. *vertex shader*) oraz fragmenty (ang. *fragment shader*). Program cieniowania wierzchołków jest wykonywany dla wszystkich wierzchołków przesłanych do karty graficznej. Głównym zadaniem jest przeliczenie współrzędnych z trójwymiarowej przestrzeni wirtualnej sceny do dwuwymiarowych współrzędnych ekranu. Program ten nie może tworzyć nowych wierzchołków. Najnowsze karty graficzne (od rodziny GeForce 8xxx) wspierają w pełni programowany strumień przetwarzania, w którym zunifikowano programowalne jednostki do jednego uniwersalnego typu. Jednostka taka może wykonywać zarówno program dla wierzchołków, jak i dla fragmentów. Nowe możliwości elastycznego budowania strumienia wizualizacji umożliwiły wprowadzenie do bibliotek *DirectX 10* i *OpenGL 3.1* kolejnego typu programu *cieniującego geometrię* (ang. *geometry shader*). Wejściem do tego programu są wyniki *składania obiektów*, zaś wyjście kierowane jest to rasteryzacji. Programy te pozwalają manipulować złożonością sceny, dodawać lub usuwać wierzchołki i mogą być używane np. do proceduralnego zagęszczania geometrii siatki. W uniwersalnych jednostkach przetwarzane mogą być również wykonywane dowolne pogramy przetwarzające dane (technologia CUDA[4] lub OpenCL[5]).

[4] Compute Unified Device Architecture, `http://www.nvidia.com/object/cuda_home_new.html`

[5] Środowisko umożliwiające tworzenie aplikacji wykorzystujących dostępne jednostki obliczeniowe CPU i GPU, `http://www.khronos.org/opencl`

Do programowania jednostek kart graficznych używa się specjalnie zaprojektowanych języków cieniowania (ang. *shader language*). Najpopularniejsze to GLSL (ang. *OpenGL Shading Language*), będący częścią standardu OpenGL oraz opracowany przez firmę nVidia język *Cg* (ang. *C for graphies*)[RK03]. Obydwa języki mają wsparcie w bibliotece VTK w postaci klasy `vtkShader`. Dzięki temu, możliwe jest użycie zaawansowanych metod obliczania np. oświetlenia oraz proceduralne generowanie tekstur. Przykładowe programy cieniujące znajdują się pod adresem `http://www.vtk.org/Wiki/VTKShaders`. Rozszerzeniem możliwości biblioteki VTK o przetwarzanie danych na poziomie procesora GPU jest projekt VTKEdge[6].

Siłą karty graficznej jest możliwość wykonywania wielu obliczeń poszczególnych etapów (przetwarzanie szeregowe) w jednym czasie (przetwarzanie równoległe). Karty mają od kilkunastu do kilkuset procesorów, które mogą realizować różne programy dotyczące geometrii (wierzchołków) lub pikseli (fragmentów).

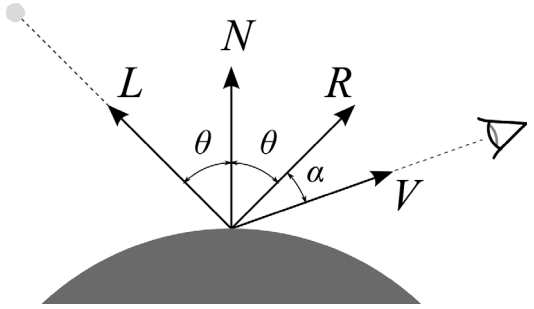
### 5.2.1. Model oświetlenia

Otrzymanie realistycznie wyglądającej powierzchni wymaga uwzględnienia oświetlenia. W grafice komputerowej można spotkać bardzo wiele różnych modeli oświetlenia, które umożliwiają generowanie zarówno obrazów bardzo realistycznych, jak również zupełnie sztucznych (np. cieniowanie naśladujące rysunek odręczny) [RK03, Zab94, rozdział 3]. To właśnie one odpowiadają za wyznaczanie kolorów poszczególnych wierzchołków oraz fragmentów obiektów trójwymiarowej sceny.

Szczególne znaczenie ma empiryczny model oświetlenia opracowany w 1975 roku przez *Phong Bui-Tuonga.* Model ten, nazwany od nazwiska twórcy *modelem oświetlenia Phonga,* dość dobrze modeluje zjawiska zwierciadlanego odbicia światła od nieidealnie gładkich powierzchni. Zakłada on, że kolor powierzchni jest wynikiem: odbicia rozproszonego, które zabarwia padające światło na kolor obiektu, oraz zwierciadlanego odbicia padającego światła (bez zmiany koloru). Model taki odpowiada np. powierzchniom plastikowym. W modelach oświetlenia uwzględnia się również współczynnik odbicia światła otaczającego, które dociera do obiektu z każdej strony. Tak więc kolor obiektu jest sumą współczynników oświetlenia: otoczenia, rozproszenia oraz rozbłysku. Każdy z tych współczynników jest zależny od kombinacji właściwości materiałowych oraz właściwości światła.

Na rysunku 5.4 przedstawiono podstawowe wektory używane do wyznaczenia koloru powierzchni z użyciem modelu oświetlenia Phonga. Wektor normalny $N$ wyznacza kierunek prostopadły do płaszczyzny stycznej do powierzchni w danym punkcie. Może on być wyznaczany analitycznie np. na podstawie współrzędnych sąsiednich wierzchołków, może być również przypisany do wierzchołka jako parametr lub do powierzchni w postaci tekstury. Wektor $L$ wyznacza kierunek od rozpatrywanego punktu powierzchni do źródła światła. Wektor $R$ jest odbitym promieniem światła $L$, zaś wektor $V$ wyznacza kierunek od analizowanego punktu obiektó do obserwatora, symbolicznie zaznaczonego okiem.

[6] Strona projektu: `http://www.vtkedge.org/`




**Rys. 5.4.** Wektory używane do wyznaczenia współczynników modelu oświetlenia Phonga: $N$ – wektor normalny, $L$ – promień światła, $R$ – promień odbity, $V$ – kierunek do obserwatora

Równanie opisujące luminancję w funkcji długości fali $L_v(\lambda)$ konkretnego punktu obiektu w modelu Phong ma następującą postać [Zab94]:

$$L_v(\lambda) = K_a(\lambda)L_a(\lambda) + K_d(\lambda)(\vec{N} \cdot \vec{L})L(\lambda) + K_s(\vec{R} \cdot \vec{L})^{N_s}L(\lambda), \qquad (5.1)$$

gdzie: $K_a$ jest współczynnikiem odbicia światła otoczenia (ang. *ambient*), $L_a$ luminancją światła otoczenia, $K_d$ to współczynnik rozproszonego odbicia światła, $L$ – luminancja źródła światła, $K_s$ – współczynnik zwierciadlanego odbicia światła, zaś $N_s$ określa współczynnik gładkości powierzchni.

W równaniu można wyróżnić trzy człony. Pierwszy odpowiada za odbicie światła otaczającego $L_a$, docierającego do obiektu ze wszystkich stron. Drugi człon zawierający współczynnik rozproszenia światła $K_d$, modeluje mikroskopową chropowatość powierzchni, na której padające światło odbija się we wszystkich kierunkach w równym stopniu. Natężenie odbitego światła jest proporcjonalne do właściwości materiału (współczynnika $K_d$), kąta padania światła na powierzchnię (iloczynu skalarnego znormalizowanych wektorów $N$ i $L$) oraz luminancji źródła światła. Ostatni człon odpowiada za modelowanie rozbłysków i zależy głównie od pozycji obserwatora względem obiektu i światła oraz parametrów powierzchni $K_s$ oraz $N_s$. Współczynnik gładkości powierzchni umożliwia regulację wielkości rozbłysku – obiekty gładkie mają bardzo mały rozbłysk (duże $N_s$). Równanie (5.1) nie uwzględnia wielu źródeł światła oraz przewiduje jedynie projekcję równoległą, a nie perspektywiczną. Rozszerzenie modelu oświetlenia Phong o kolorowe rozbłyski $K_s(\lambda)$ i projekcję perspektywiczną ma następującą postać:

$$\begin{aligned} L_v(\lambda) &= K_a(\lambda)L_a(\lambda) + K_d(\lambda)\sum_{n=1}^{l_s}(\vec{N} \cdot \vec{L_n})L_n(\lambda) + \\ &+ K_s(\lambda)\sum_{n=1}^{l_s}(\vec{N} \cdot \vec{H_n})^{N_s}L_n(\lambda) \end{aligned} \qquad (5.2)$$

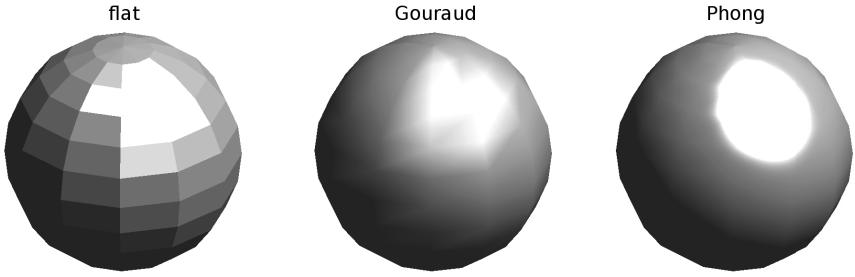
Występujący we wzorze wektor $H$ leży na dwusiecznej kąta tworzonego przez wektor światła $L$ i wektor obserwatora $V$. Odpowiada on wektorowi normalnemu płaszczyzny idealnego odbicia. Iloczyn skalarnych wektorów $N$ oraz $H$ uwzględnia projekcję perspekty-

wiczną. Dla wszystkich świateł występujących w scenie ($l_s$) wyznaczane są składowe odpowiedzialne za światło rozproszone oraz światło odbite. Model ten może być również uzupełniony o efekt samoświecenia powierzchni, który realizowany jest przez dodanie do równania (5.2) stałej wartości.

Równanie (5.2) stosuje się zazwyczaj do obliczenia trzech składowych koloru: R, G i B. Również parametry materiałowe $K_a$, $K_d$, $K_s$ określane są jako trójelementowe wektory, zawierające współczynniki dla składowych modelu RGB.

## 5.2.2. Interpolacja kolorów

Ze względu na małe koszty obliczeniowe, model oświetlenia Phonga został zaimplementowany w kartach graficznych do wyznaczania koloru wierzchołków. W zależności od przyjętego algorytmu interpolacji kolorów, możliwe jest sterowanie wydajnością oraz jakością generowanych obrazów. Na rysunku 5.5 przedstawiono wyniki działania trzech podstawowych algorytmów interpolacji oświetlenia. Można zauważyć, że uzyskiwane obrazy sfery znacznie się różnią.




**Rys. 5.5.** Trzy podstawowe metody interpolacji koloru powierzchni wielokątów

Najprostszą metodą interpolacji kolorów jest cieniowanie płaskie *flat.* W metodzie tej kolor wielokąta wyznacza się jednokrotnie, na podstawie: modelu oświetlenia Phonga, właściwości materiałowych, kierunku oświetlenia oraz wartości wektora normalnego wielokąta. Wszystkim fragmentom wielokąta przypisuje się tylko jeden, wcześniej wyliczony kolor. Jak widać, jest to metoda bardzo szybka, jednocześnie dająca najgorsze wyniki. Granice między elementami powierzchni są szczególnie widoczne (efekt pasm Macha), utrudniona jest również percepcja kształtu.

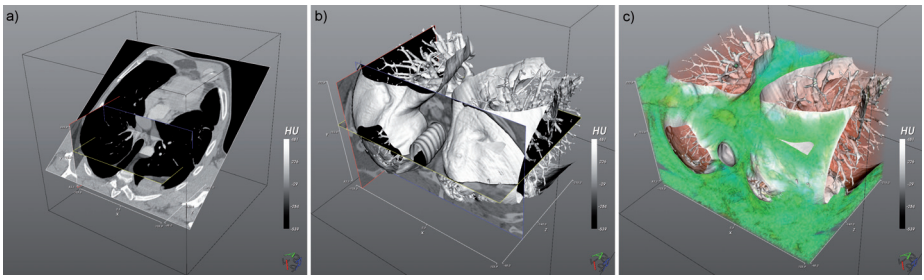
Algorytm interpolacji Gouraud oblicza kolory dla wszystkich wierzchołków, a następnie dla poszczególnych fragmentów dokonuje liniowej interpolacji wartości składowych koloru (w pierwotnej wersji była to interpolacja jasności). Algorytm ten umożliwia zachowanie płynnych przejść między kolorami sąsiadujących ze sobą wielokątów, przez co znacznie poprawia percepcję kształtu. Niestety, taki algorytm interpolacji traci informację o rozbłyskach, które nie obejmują wierzchołków, ponieważ kolor obliczany z modelu oświetlenia Phonga wyznaczany jest tylko w wierzchołkach. Problem ten można ograniczyć, stosując gęste siatki opisujące kształt.

Najbardziej zaawansowanym algorytmem jest interpolacja Phonga, w którym dla wszystkich fragmentów wyznaczany jest kolor na podstawie modelu oświetlenia Phonga oraz interpolowanego wektora normalnego. W tym przypadku dla poszczególnych fragmentów nie jest interpolowany kolor tylko wektor normalny. Równanie modelu oświetlenia Phonga (5.2) obliczane jest dla wszystkich fragmentów. Dzięki temu, rozbłyski obliczane są bardzo dokładnie nawet na dużych wielokątach. Odbywa się to jednak kosztem bardzo dużej liczby obliczeń. Z tego powodu interpolacja Phonga nie była realizowana sprzętowo. Obecnie model oświetlenia Phonga wraz z interpolacją Phonga może być realizowany w postaci programu *shadera*.

## 5.3. Rendering objętościowy (wolumetryczny)

Opisany w podrozdziale 4.4.2 algorytm konturowania *marching cubes* umożliwia wizualizację przestrzennych danych skalarnych poprzez rekonstrukcję izopowierzchni odpowiadającej np. powierzchni skóry, kości lub innych tkanek. Taka prezentacja danych ma jednak poważną wadę – przedstawia jedynie fragment danych, np. jedną warstwę danych odpowiadającą zadanym z góry wartościom. Również stosowanie płaszczyzn przecinających dane wolumetryczne, a następnie mapowanie wartości skalarnych w postaci koloru, prezentuje jedynie informację lokalną, tylko w miejscu przecięcia. Metody te, zwane również *pośrednim renderingiem wolumetrycznym* (ang. *indirect volume rendering*), używają pośrednich reprezentacji do prezentacji danych. Oczywiście, możliwe jest wygenerowanie wielu konturów czy też użycie wielu płaszczyzn przecinających dane, jednak nadal mogą to być rozwiązania nieakceptowalne, utrudniające, a nie ułatwiające analizę danych objętościowych.

Rendering wolumetryczny (ang. *direct volume rendering*) jest rozwiązaniem problemu wizualizacji danych objętościowych, który umożliwia prezentowanie danych w całości, bez ograniczenia do konturów (izopowierzchni) czy też przecięć. W metodach tych dane reprezentowane są na otrzymywanym obrazie w sposób bezpośredni. Na rysunku 5.6 przedstawiono porównanie możliwości wizualizacji trzech podstawowych metod: mapowania danych na dwuwymiarową płaszczyznę przecinającą dane (ang. MPR – *Multi Planar Reformation*), kontur (izopowierzchnia) oraz rendering wolumetryczny tkanek miękkich. Rysunek przedstawia wizualizację danych pochodzących z tomografii komputerowej – fragment klatki piersiowej.




**Rys. 5.6.** Trzy podstawowe tryby wizualizacji danych objętościowych na przykładzie danych z tomografii komputerowej: a) przecięcie płaszczyzną (MPR), z mapowaniem kolorów, b) wyznaczenie konturu (na poziomie powietrze-tkanka) oraz c) rendering wolumetryczny

Rendering wolumetryczny umożliwia prezentację wszystkich danych na jednym obrazie, będącym wynikiem przypisania wartościom skalarnym koloru oraz przeźroczystości. Dodatkowo, możliwe jest również uwzględnienie oświetlenia (zazwyczaj model oświetlenia Phonga), co może znacząco poprawić percepcję kształtu. Rendering wolumetryczny przedstawia dane w postaci „kolorowych chmur", dzięki temu umożliwia patrzenie „przez dane". Jest to szczególnie ważne w przypadku danych medycznych, np. z tomografii komputerowej, w których każdy woksel niesie cenną informację. Możliwość zauważenia drobnych różnic w wartościach (np. gęstości tkanek) może decydować o wyniku diagnozy stawianej na podstawie wizualizacji.

Zadaniem renderingu jest wygenerowani obrazu, przedstawiającego wzajemne oddziaływanie między obiektami wirtualnej sceny: światłem, obiektami oraz przestrzenią, w której się znajdują (np. uwzględniając zjawisko mgły) [PB07]. Oddziaływania te można opisać takimi zjawiskami, jak: absorpcja, emisja oraz rozproszenie światła (ang. *transport theory of light* opisał Krüger w pracy [Kru90]). W przypadku renderingu wolumetrycznego można się jednak ograniczyć do prostszego modelu uwzględniającego jedynie emisję oraz absorpcję (ang. *density emitter model*), w którym każdy element danych objętościowych (każda wartość skalarna) traktowana jest jako źródło światła o bardzo małych rozmiarach, którego jasność maleje wraz z odległością [PB07]. Możliwe są również modyfikacje tego modelu polegające na uwzględnianiu tylko emisji lub tylko absorpcji.

Emisja poszczególnych „źródeł światła" oznaczana jest symbolem $Q_\lambda(s)$, gdzie $\lambda$ oznacza długość emitowanej fali świetlnej, zaś $s$ kierunek wyznaczany przez promień przechodzący przez dane. Promień ten biegnie z nieskończoności przez punkt w przestrzeni odpowiadający wartości skalarnej, przez płaszczyznę obrazu do obserwatora. Dla uproszczenia zapisu, opuszcza się w oznaczeniu $\lambda$ zakładając, że funkcja dotyczy jednej długości fali światła (zazwyczaj składowej modelu RGB). Intensywność światła, docierająca do konkretnego punktu obrazu i wywołana emisją danych, znajdujących się na drodze promienia $S$, opisana jest następującą zależnością:

$$I(s) = I_{s_0} + \int_{s_0}^{s} Q(t)dt = I_{s_0} + \int_{s_0}^{s} T_F(v(t))dt, \tag{5.3}$$

gdzie: $I_{S_0}$ jest początkową intensywnością światła (światło tła), $s_0$ jest punktem, w którym promień $S$ przecina dane po raz pierwszy (promień „wchodzi do danych"), zaś $s$ jest puntkem, w którym promień „opuszcza dane".

Emisyjność $Q(s)$ zastępuje się *funkcją przejścia* $T_F$ wyznaczaną dla konkretnych wartości skalarnych $v(s)$ w punkcie $s$. Możliwe jest również uwzględnienie oświetlenia w punkcie $s$. Dla takiego punktu, poza obliczoną wartości funkcji przejścia, która wyznacza kolor punktu, oblicza się również wektor normalny (np. na podstawie gradientu sąsiednich wartości skalarnych). Na podstawie koloru punktu, wektora normalnego, parametrów materiałowych oraz pozycji światła obliczana jest nowa wartość koloru punktu, odpowiadająca modelowi oświetlenia Phonga.

Absorpcja modeluje zanik intensywności światła spowodowany pochłanianiem ośrodka, w którym dane są umieszczone (przestrzeń modelowana jest jako jednorodna mgła). Równanie opisujące zmianę intensywności światła na drodze promienia $S$ ma postać:

$$I(s) = I_{s_0} \cdot e^{\left(-\int_{s_0}^{s} \tau(t)dt\right)}, \tag{5.4}$$

gdzie parametr $\tau(s)$ opisuje współczynnik tłumienia światła w punkcie $s$ i odpowiada za przejrzystość. Jest on wartością komplementarną do współczynnika $\alpha$ w modelu RGBA: $\alpha(s) = 1 - \tau(s)$, $\tau \in [0..1]$.

Rendering wolumetryczny uwzględniający zarówno zjawisko emisyjności, jak i absorpcji światła, opisany jest tzw. równaniem renderingu wolumetrycznego (ang. *volume rendering equation*):

$$\begin{aligned} I(s) &= I_{s_0} \cdot e^{\left(-\int_{s_0}^{s} \tau(t)dt\right)} + \int_{s_0}^{s} Q(p) \cdot e^{\left(-\int_{s_0}^{s} \tau(t)dt\right)} dp \\ &= I_{s_0} \cdot T_{s_0}(s) + \int_{s_0}^{s} Q(p) \cdot T_p(p)dp, \end{aligned} \tag{5.5}$$

gdzie $T_x(.)$ jest *funkcją osłabienia.*

Niestety, równania całkowe nie mogą być wyznaczane w sposób analityczny dla dowolnych funkcji przejścia $T_F()$, funkcji osłabienia $T()$ oraz niezbędnych interpolacji dyskretnych danych. Dlatego, równanie całkowe 5.5 dyskretyzuje się poprzez podział promienia $S$ na $k$ odcinków oddalonych o $\Delta s$ [PB07]:

$$I(s) = I_0 \prod_{k=0}^{n-1} t_k + \sum_{k=0}^{n-1} Q(k \cdot \Delta s) \cdot \Delta s \prod_{j=k+1}^{n-1} t_j, \tag{5.6}$$

gdzie $t_k$ jest przeźroczystością w punkcie $k$, zaś człon iloczynu odpowiada dyskretnej wersji funkcji osłabienia:

$$T_0(s) = e^{\left(-\int_{s_0}^{s} \tau(t)dt\right)} = \prod_{k=0}^{n-1} e^{-\tau(k\Delta t)\Delta t} = \prod_{k=0}^{n-1} t_k. \tag{5.7}$$

Równanie (5.6) opisuje w jaki sposób promień $S$ przechodząc przez dane akumuluje $k$ wartości intensywności. Wynikają one z intensywności źródeł $Q_k$ osłabionych iloczynem przeźroczystości $t_j$.

### 5.3.1. Strumień wizualizacji wolumetrycznej

Zadaniem strumienia wizualizacji wolumetrycznej jest wykonanie niezbędnych obliczeń, umożliwiających wyznaczenie intensywności opisanej równaniem (5.6). Można wyróżnić trzy podstawowe kroki w strumieniu wizualizacji wolumetrycznej[PB07]:

**Próbkowanie** jest operacją polegającą na wyznaczeniu równania prostej reprezentującej promień S przecinający dane. Na podstawie tego równania, wyznaczane są punkty przecięcia promienia z danymi $k \cdot \Delta s = f(x, y, z)$. Dla tych punktów w przestrzeni, przez interpolację danych, wyznaczane są wartości skalarne $v(k)$. Sposób interpolacji ma bardzo duży wpływ na wyniki wizualizacji.

**Klasyfikacja** oraz **wyznaczenie oświetlenia**. Dla punktów, otrzymanych w wyniku próbkowania, wyznaczana jest wartość $Q_k$, na podstawie klasyfikacji wartości $v(k)$ przy użyciu *funkcji przejścia* $Q_k = T_F(v(k))$. W wyniku klasyfikacji otrzymuje się wartości składowych koloru (RGB). Kolor ten może być użyty bezpośrednio w równaniu (5.6), może być również zmodyfikowany poprzez uwzględnienie oświetlenia. W celu obliczenia wektora normalnego, niezbędnego do obliczeń w modelu oświetlenia Phong, wyznaczany jest gradient danych skalarnych. Podczas klasyfikacji określane są również wartości współczynników przeźroczystości $t_j$ na podstawie *funkcji nieprzeźroczystości* OTF (ang. *opacity transfer function*). Dzięki tej funkcji możliwe jest „wyłączenie" z wizualizacji konkretnych danych poprzez przypisanie im pełnej przeźroczystości.

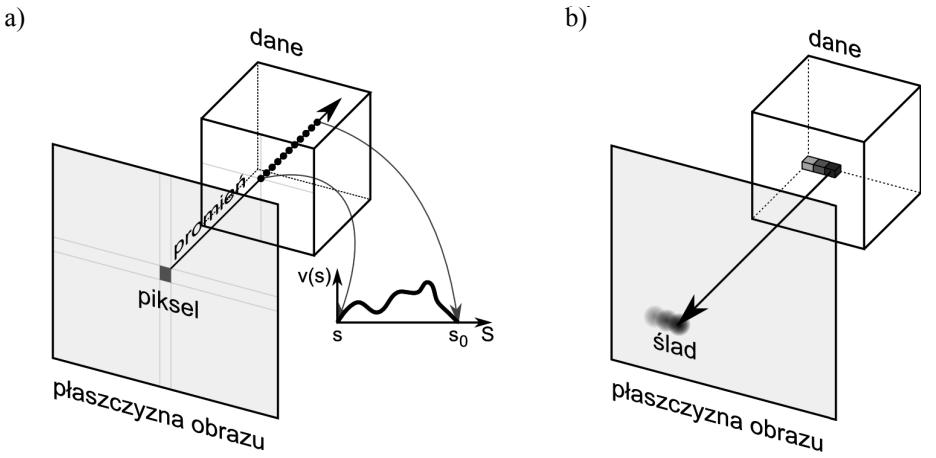
**Składanie obrazu** jest odpowiednikiem całkowania wzdłuż promienia i polega na obliczeniu wartości intensywności zgodnie z równaniem (5.6). „Składanie" wartości intensywności na podstawie $k$ punktów może odbywać się w dwóch kierunkach: od danych do obrazu (ang. *back to front*) oraz od obrazu do danych (ang. *front to back*). Obydwie metody mają swoje zalety oraz wady. Możliwe jest również zastosowanie zupełnie innej funkcji składającej, np. przypisującej intensywności $I(s)$ wartość maksymalną wzdłuż całego promienia (ang. MIP – *Maximum Intensity Projection*), pierwszą lokalną wartość maksymalną (ang. CVP – *Closest Vassel Projection*), wartość średnią lub też dokonującej sumowania wszystkich wartości (symulacja zdjęcia rentgenowskiego).

Różnice między algorytmami wizualizacji wolumetrycznej można sprowadzić do różnic w kolejności przeprowadzania podstawowych kroków strumienia lub w zastosowaniu różnych metod (np. różnych algorytmów interpolacji danych). Jeżeli najpierw wykonywane jest próbkowanie, a następnie klasyfikacja, wówczas taki algorytm nazywa się *post-classified (post-shaded) volume rendering*. Jeżeli te operacje są zamienione miejscami w strumieniu wizualizacji (najpierw jest klasyfikacja), wówczas algorytm nazywany jest *pre-classified volume rendering*.

## 5.3.2. Podstawowe algorytmy

Algorytmy renderingu wolumetrycznego można podzielić na dwie grupy, różniące się sposobem generowania obrazu: metody zorientowanie na obraz (ang. *image-order*) oraz na obiekty (ang. *object-order*). W pierwszej grupie, obraz powstaje przez kolejne wyznaczanie wartości poszczególnych pikseli. Wartość koloru pikseli jest wyznaczana jednokrotnie, od pierwszego do ostatniego piksela. Algorytmy drugiej grupy generują obraz poszczególnych obiektów sceny od najbardziej oddalonego do najbliższego. Metody *object-order* podobne są do tworzenia wyklejanki z papieru – najpierw przyklejany jest do tła prostokąt tworzący ścianę budynku, potem na tło i ścianę przyklejany jest trapez symbolizujący dach, później okna, drzwi, płotek przed domem, itd. Wartość koloru konkretnego piksela może więc być modyfikowana wielokrotnie.

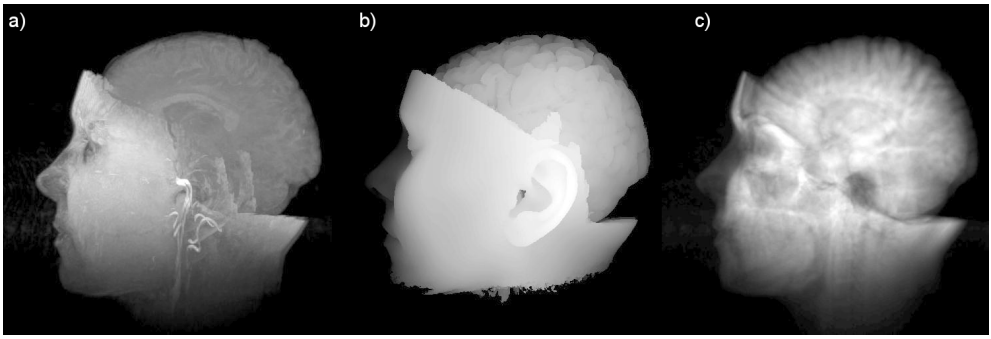
Rysunek 5.7 przedstawia zasadę tworzenia obrazów metodami zorientowanymi na obraz 5.7a) oraz metodą zorientowaną na obiekty 5.7b).




**Rys. 5.7.** Powstawanie obrazu za pomocą renderingu wolumetrycznego: a) metodą śledzenia promieni (algorytm typu *image-order*) oraz metodą *splatting* (algorytm typu *object-order*)

**Metoda śledzenia promieni**

Algorytm *śledzenia promieni* (ang. *ray casting*) jest podstawowym algorytmem wolumetrycznej wizualizacji danych. Jest to algorytm bezpośrednio realizujący dyskretne równanie renderingu wolumetrycznego (równanie (5.6)). Schemat działania tej metody przedstawiono na rysunku 5.7a. W metodzie tej, przez wizualizowane dane prowadzone są *promienie*, które biegną od oka przez kolejne piksele obrazu w kierunku danych. Liczba promieni równa jest liczbie pikseli obrazu (jest to algorytm typu *image-order*). W celu poprawy jakości obrazu oraz spełnienia założeń *twierdzenia o próbkowaniu* stosuje się nadpróbkowanie – dla jednego piksela generowane jest cztery (2 × 2) lub więcej promieni. Niestety, metoda ta jest bardzo kosztowna obliczeniowo. Metoda *ray casting* jest uproszczeniem metody *ray tracing*, w której poza głównym promieniem (od obrazu do danych) rozpatruje się również dodatkowe promienie odbite oraz załamane. Jeżeli promień przecina dane, następuje próbkowanie i interpolacja danych. Na rysunku 5.7a miejsca próbkowania zaznaczono punktami. Położenie punktów zależy od przyjętego kroku próbkowania ($\Delta s$) oraz kierunku pod jakim promień przecina dane. Następnie dokonywana jest interpolacja wartości skalarnych $v(s)$ leżących na promieniu, przedstawiona na rysunku w postaci wykresu. Na podstawie tych wartości dokonywana jest klasyfikacja oraz składanie. W wyniku obliczenia wartości prawej strony równania (5.6) otrzymywany jest kolor konkretnego piksela obrazu. Rysunek 5.8 przedstawia wyniki wizualizacji wolumetrycznej wykonanej metodą śledzenia promieni, w której wykorzystano trzy różne funkcje składające obraz. W pierwszym przypadku 5.8a jasność piksela zależy od największej wartości znajdującej się na promieniu. Jest to technika szczególnie przydatna do wizualizacji danych z angiografii, w której badane są naczynia krwionośne. W drugim przypadku 5.8b jasność została uzależniona od odległości do konkretnej wartości skalarnej, liczonej wzdłuż promienia (kolor biały to odległość najmniejsza). Ostatni przykład prezentuje wartość średnią, obliczoną z danych leżących na promieniu.




**Rys. 5.8.** Przykład zastosowania trzech różnych metod składania obrazu algorytmu śledzenia promieni: a) wizualizacja wartości maksymalnej (metoda MIP) znajdującej się na promieniu, b) odległość do stałej wartości, c) wizualizacja wartości średniej [SML04]

**Algorytm *shear warp***

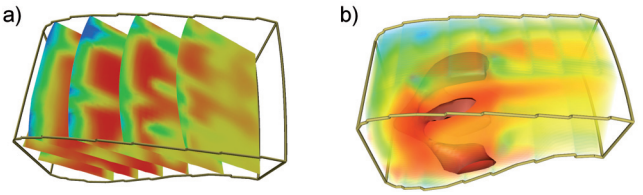
Jest to bardzo wydajna wersja prostego algorytmu śledzenia promieni, opracowana w 1994 roku przez Philippe Lacroute'a i Marca Levoya. Głównym założeniem algorytmu było uproszczenie etapu próbkowania. Algorytm ten nie może być zaliczony do grona algorytmów typu *image-based*, ponieważ zamiast śledzenia wielu promieni pod różnymi kątami (projekcja perspektywiczna) kolejne warstwy danych są przesuwane względem siebie (ang. *shear*), skalowane, a następnie prostopadle rzutowane na pierwszą warstwę. Tak otrzymany obraz jest następnie wyginany (ang. *warp*) w celu uwzględnienia kierunku patrzenia na dane. Algorytm ten jest przykładem połączenia metod *image-order* oraz *object-order*. Szczegółowy opis znajduje się na stronie: `http://graphics.stanford.edu/papers/shear/`.

**Algorytm *splatting***

Przykładem algorytmy zorientowanego na obiekty jest algorytm *splatting*, który dokonuje projekcji kolejnych wokseli danych na obraz. Schemat działania tej metody przedstawiono na rysunku 5.7b. Przed projekcją dane (skalarne wartości odpowiadające wokselom) są klasyfikowane: obliczany jest kolor oraz przezroczystość punktu. Następnie dokonywany jest splot punktu z funkcją rozmywającą, zazwyczaj dwuwymiarową funkcją Gausa. Rozmiar rozmycia dobiera się w zależności od rozdzielczości generowanego obrazu oraz odległości między punktami danych po projekcji. Dobiera się go w taki sposób, by „ślady" zostawiane przez kolejne woksele zachodziły na siebie, tworząc ciągłą powierzchnię. Efekt rozmycia szczegółów obrazu można ograniczyć stosując przetwarzanie w schemacie *post-classified.* Wydajność algorytmu jest w mniejszym stopniu zależna od rozmiarów generowanego obrazu, niż ma to miejsce w przypadku algorytmów *image-order* takich, jak śledzenie promieni. Ze względu na prostotę algorytmu *spatting* oraz możliwość jego zrównoleglenia, został on zaimplementowany w postaci programów wykonywanych przez procesory karty graficznej GPU [CRZP04].

**Algorytmy wykorzystujące sprzętowe mapowanie tekstur**

Przykładem wykorzystania możliwości sprzętowego mapowania kolorów karty graficznej są algorytmy nazywane *texture mapping*. Rysunek 5.9 przedstawia ideę działania takiego algorytmu. W metodzie tej dane przecinane są dużą liczbę równoległych płaszczyzn, na których mapuje się skalarne wartości danych za pomocą funkcji przejścia (ang. *transfer function*). Wartościom danych przypisuje się kolor oraz przeźroczystość wynikające z przyjętego algorytmu klasyfikacji. Do mapowania kolorów używa się tekstur dwuwymiarowych (w starszym sprzęcie) i trójwymiarowych oraz sprzętowych mechanizmów karty graficznej, umożliwiających zmiennoprzecinkowe adresowanie pikseli tekstury oraz interpolację danych.



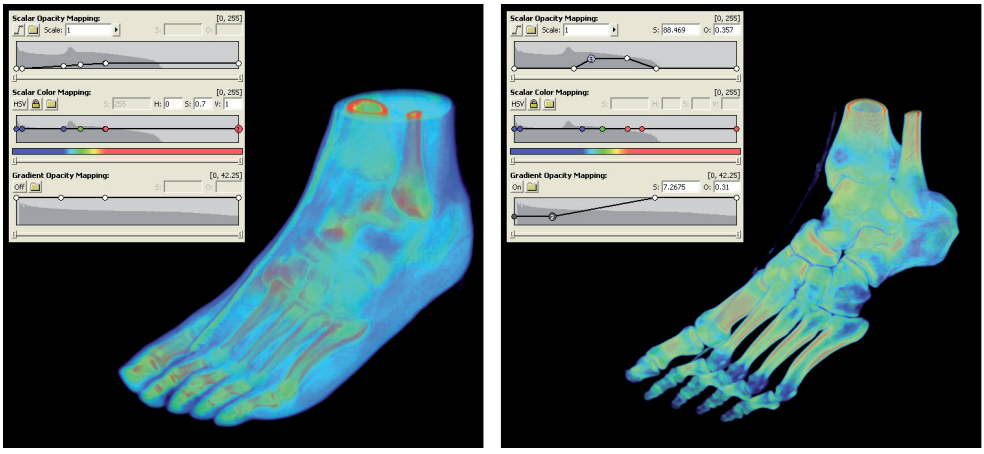

**Rys. 5.9.** Zasada działania algorytmu *texture mapping* [PB07]

Na rysunku 5.9a przedstawiono przykład małej liczby płaszczyzn przecinających, z przypisaną jedynie funkcją przejścia, bez uwzględnienia przeźroczystości. Natomiast rysunek 5.9b przedstawia wynik działania właściwego algorytmu renderingu wolumetrycznego (duża liczba płaszczyzn) z przypisaną funkcją przeźroczystości oraz wyliczoną izopowierzchnią (renderingu oświetlonej powierzchni). Jak widać, dzięki wykorzystaniu pośrednich obiektów graficznych (płaszczyzny z nałożonymi teksturami) możliwe jest proste i jednocześnie wydajne łączenie renderingu wolumetrycznego oraz innych technik pośredniego renderingu objętościowego (np. izopowierzchni). Wadą tej metody są artefakty występujące na krawędziach oraz znaczące błędy przy wizualizacji z użyciem „kamery szerokokątnej", powstające na skutek różnej odległości między warstwami widzianymi pod ostrym kątem.

## 5.3.3. Funkcja przejścia

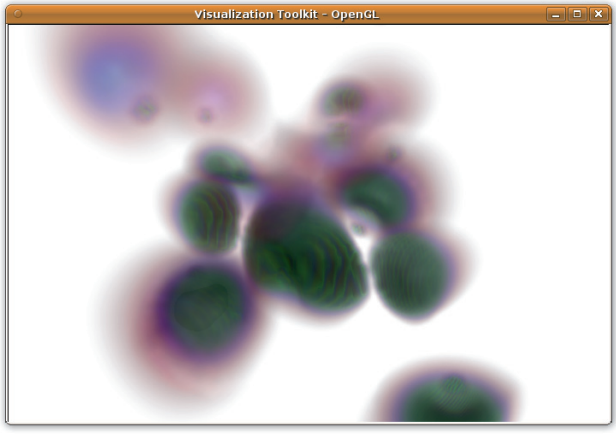
Dobór funkcji przejścia (ang. *transfer function*) jest podstawowym zagadnieniem w każdym algorytmie renderingu wolumetrycznego. To na podstawie tej funkcji konkretnym wartościom przypisywane są kolory oraz przeźroczystość. Od niej głownie zależy wynik wizualizacji. Na rysunku 5.10 przedstawiono przykład wizualizacji tych samych danych przy użyciu różnych zestawów funkcji przejścia (ang. *Scalar Color Mapping*), funkcji przezroczystości (ang. *Scalar Opacity Mapping*) oraz z uwzględnieniem gradientu danych (ang. *Gradient Opacity Maping*). Wykorzystane funkcje przejścia definiowane są w postaci funkcji jednowymiarowych przy pomocy kilku *wartości węzłowych*. Jest to metoda najpopularniejsza, prosta w implementacji oraz łatwa w zrozumieniu. Dla konkretnych wartości wizualizowanych danych (skalarów) przypisuje się albo wartości skalarne (współczynnik przeźroczy-

stości) lub też wektorowe (składowe barwy, zapisane w konkretnym modelu barwnym, np. RGB lub HSV). Podczas klasyfikacji wartościom danych przypisuje się kolory wynikające z interpolacji funkcji przejścia obliczanej na podstawie wartości w węzłach.




**Rys. 5.10.** Przykład zastosowania różnych funkcji przejścia do wizualizacji danych medycznych

Wartości węzłowe mogą być ustalone w sposób arbitralny przez użytkownika (metodą „prób i błędów"), mogą również być wspomagane np. poprzez przedstawienie histogramu danych (zarówno globalnego, jak i lokalnego). Dogodnym miejscem zdefiniowania wartości węzłowej dla funkcji przejścia są ekstrema lokalne występujące na wykresie histogramu (rysunek 5.10).




**Rys. 5.11.** Wynik działania programu `SimpleRayCast.tcl`

Oprócz jednowymiarowych funkcji przejścia stosuje się również funkcje wielowymiarowo, które poza lokalną wartością skalarną danego punktu uwzględniają również inne informacje podczas klasyfikacji [PB07]. Wzięte pod uwagę mogą być: wartość gradientu danych, aproksymowana druga pochodna, lokalna krzywizna powierzchni czy też odległość do innego obiektu referencyjnego.

## 5.3.4. Przykład programu

Listing 5.1 zawiera program (`/usr/share/vtk/VolumeRendering/Tcl/SimpleRayCast.tcl`), który wizualizuje prawdopodobieństwo wystąpienia elektronu wokół cząsteczki. Wynik działania programu przedstawiono na rysunku 5.11.

**Listing 5.1:** Przykład programu w języku Tcl generującego obraz wolumetryczny metodą *ray cast*

```
# wczytanie niezbędnych bibliotek
package require vtk
package require vtkinteraction

# utworzenie okna wizualizacji
vtkRenderer ren1
vtkRenderWindow renWin
    renWin AddRenderer ren1
vtkRenderWindowInteractor iren
    iren SetRenderWindow renWin

# wczytanie danych z pliku
vtkStructuredPointsReader reader
    reader SetFileName "$VTK_DATA_ROOT/Data/ironProt.vtk"

# utworzenie funkcji przejścia odpowiedzialnej za przeźroczystość
vtkPiecewiseFunction opacityTransferFunction
    opacityTransferFunction AddPoint  20   0.0
    opacityTransferFunction AddPoint  255  0.2

# utworzenie funkcji przejścia odpowiedzialnej za kolor punktów
vtkColorTransferFunction colorTransferFunction
    colorTransferFunction AddRGBPoint      0.0 0.0 0.0 0.0
    colorTransferFunction AddRGBPoint     64.0 1.0 0.0 0.0
    colorTransferFunction AddRGBPoint    128.0 0.0 0.0 1.0
    colorTransferFunction AddRGBPoint    192.0 0.0 1.0 0.0
    colorTransferFunction AddRGBPoint    255.0 0.0 0.2 0.0

# utworzenie obiektu opisującego wygląd danych
vtkVolumeProperty volumeProperty
    volumeProperty SetColor colorTransferFunction           # przypisanie mapowania koloru
    volumeProperty SetScalarOpacity opacityTransferFunction # przypiasnie mapowania
      przeźroczystości
    volumeProperty ShadeOn                                  # włączenie cieniowania
    volumeProperty SetInterpolationTypeToLinear             # włączenie interpolacji liniowej

# obiekt odpowiedzialny za składanie obrazu, mapujący dane, dostępne są również:
#   vtkVolumeRayCastMIPFunction - generujący obrazy maksymalnej intensywności (MIP)
#   vtkVolumeRayCastIsosurfaceFunction - generujący obraz izopowierzchni
vtkVolumeRayCastCompositeFunction compositeFunction
```

```
40 vtkVolumeRayCastMapper volumeMapper
41     volumeMapper SetVolumeRayCastFunction compositeFunction
42     volumeMapper SetInputConnection [reader GetOutputPort]
43
44 # obiekt łączący funkcję mapującą z parametrami, umożliwia określenie położenia danych na scenie
45 vtkVolume volume
46     volume SetMapper volumeMapper
47     volume SetProperty volumeProperty
48
49 ren1 AddVolume volume           # dodanie aktora do sceny
50 ren1 SetBackground 1 1 1        # konfiguracja koloru tła
51 renWin SetSize 600 600         # wymiary okna wizualizacji
52 renWin Render                  # rozpoczęcie wizualizacji
53
54 # funkcja pomocnicza, odpowiedzialna za zakończenie programu
55 proc TkCheckAbort {} {
56   set foo [renWin GetEventPending]
57   if {$foo != 0} {renWin SetAbortRender 1}
58 }
59 renWin AddObserver AbortCheckEvent {TkCheckAbort} # dodanie obserwatora
60
61 # konfiguracja obiektu interaktora, odpowiedzialnego za interaktywne sterowanie wizualizacji
62 iren AddObserver UserEvent {wm deiconify .vtkInteract}
63 iren Initialize
64 wm withdraw .
```

Algorytm *ray cast* zaimplementowany w obiekcie `vtkVolumeRayCastMapper` można zmienić na inny, np. wykorzystujący sprzętowe mapowanie tekstur, zmieniając obiekt mapujący (linia 40) na `vtkVolumeTextureMapper2D` lub `vtkVolumeTextureMapper3D`. Możliwe jest również użycie klasy `vtkVolumeProMapper`, która udostępnia interfejs do renderingu przy użyciu sprzętowej karty *VolumePRO*[7]. Istnieje również zamiennik klasy `vtkVolumeRayCastMapper` w postaci specjalizowanej wersji `vtkFixedPointVolumeRayCastMapper`, która posiada większe możliwości konfiguracji. Klasa `vtkGPUVolumeRayCastMapper` dostępna w VTK od wersji 5.6, udostępnia implementację algorytmu *ray cast* obliczaną na procesorze GPU karty graficznej.

W przedstawionym programie występują dwa obiekty, które definiują jednowymiarowe funkcje przejścia: `vtkPiecewiseFunction` (17) określa przeźroczystość, zaś `vtkColorTransferFunction` (22) kolor, jaki będzie przypisany do wartości liczbowych. Zastosowane funkcje przejścia używają interpolacji liniowej. Możliwe jest również użycie obiektów `vtkLogLookupTable`, który udostępnia interpolację wartości skalarnych z użyciem skali logarytmicznej o podstawie 10. Klasa `vtkWindowLevelLookupTable` umożliwia wygodne określenie funkcji przejścia przez podanie wartości środkowej (ang. *level*) oraz szerokości okna (ang. *window*). Tak zdefiniowana funkcja przejścia ma postać liniowo narastającej „rampy". W przypadku wizualizacji danych o charakterze dyskretnym, np. wyników segmentacji, w których kolejne wartości całkowite odpowiadają indeksom obiektów, pomocna może być funkcja `vtkDiscretizableColorTransferFunction`.

[7] Więcej informacji można znaleźć na stronie: `http://www.terarecon.com/`

## 5.4. Podsumowanie

W niniejszym rozdziale zasygnalizowane zostały jedynie zupełne podstawy zaawansowanych metod wizualizacji. Część bardzo ważnych zagadnień, takich jak np. metody tworzenia wielowymiarowych funkcji przejścia, jest obecnie przedmiotem badań naukowych. Poruszana tematyka jest bardzo obszerna, dlatego osobom zainteresowanym poszerzeniem wiedzy na temat wizualizacji w medycynie szczególnie polecam książkę *Visualization in Medicine*[PB07], która jest znakomitym kompendium wiedzy z bardzo licznymi przykładami praktycznych aplikacji. Również pozycja *Trends in Interactive Visualization*[ZSAL08] może być bardzo cennym źródłem informacji.

# Bibliografia


[AbKI04] Authored and Published by Kitware Inc., editors. *The VTK User's Guide.* Kitware Inc., 2004.

[BDD+07] J. Bułat, K. Duda, M. Duplaga, R. Fraczek, A. Skalski, M. Socha, P. Turcza, T.P. Zieliński. Data processing tasks in wireless GI endoscopy: Image-based capsule localization; navigation and video compression. In *Engineering in Medicine and Biology Society, 2007. EMBS 2007. 29th Annual International Conference of the IEEE*, pages 2815–2818, 22–26 2007.

[Ber83] Jacques Bertin. *Semiology of graphics: diagrams, networks, maps / Jacques Bertin; translated by William J. Berg.* University of Wisconsin Press, Madison, Wis., 1983.

[CRZP04] Wei Chen, Liu Ren, Matthias Zwicker, Hanspeter Pfister. Hardware-accelerated adaptive EWA volume splatting. In *IEEE VISUALIZATION*, pages 67–74. IEEE Computer Society, 2004.

[DKSSW06] Lidia Drabik, Aleksandra Kubik-Sokół, Elżbieta Sobol, Lidia Wiśniakowska. *Słownik języka polskiego.* Wydawnictwo Naukowe PWN, 2006.

[Kru90] Wolfgang Krueger. The application of transport theory to visualization of 3D scalar data fields. In *VIS '90: Proceedings of the 1st conference on Visualization '90,* pages 273–280, Los Alamitos, CA, USA, 1990. IEEE Computer Society Press.

[LC87] William E. Lorensen, Harvey E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph.,* 21(4):163–169, 1987.

[Mac86] Jock Mackinlay. Automating the design of graphical presentations of relational information. *ACM Trans. Graph.*, 5(2):110–141, 1986.

[MJM+10] Duplaga Mariusz, Bułat Jarosław, Leszczuk Mikołaj, Socha Mirosław, Romaniak Piotr, Turcza Paweł. *The BRONCHOVID – computer system supporting bronchoscopy laboratory*, volume 2 of *Advances in Intelligent and Soft Computing.* Springer-Verlag, 2010.

[Ost35] G. Osterberg. *Topography of the layer of rods and cones in the human retina.* Acta Ophthalmol., 1935.

[PB07] Bernhard Preim, Dirk Bartz. *Visualization in Medicine: Theory, Algorithms, and Applications.* The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007.

[Ray] Kryterium Rayleigha. `http://pl.wikipedia.org/wiki/Kryterium_Rayleigha.`

[RK03] Fernando Randima, Mark J. Kilgard. *Język Cg. Programowanie grafiki w czasie rzeczywistym.* Helion, 2003.

[Rob99] Jonathan C. Roberts. Display models for visualization. *Information Visualisation 1999 IEEE, International Conference on*, pages 200–206, 1999.

[SML04] Will Schroeder, Ken Martin, Bill Lorensen. *The Visualization Toolkit, Third Edition.* Kitware Inc., 2004.

[SSZD07] Anadrzej Skalski, Mirosław Socha, Tomasz Zieliński, Mariusz Duplaga. Colon cleansing for virtual colonoscopy using non-linear transfer function and morphological operations. In *Imaging Systems and Techniques, 2007. IST '07. IEEE International Workshop on*, pages 1–5, 5–5 2007.

[TZD+06] Tomasz Twardowski, Tomasz Zieliński, Krzysztof Duda, Mirosław Socha, Mariusz Duplaga. Fast estimation of broncho-fiberoscope egomotion for CT-guided transbronchial biopsy. In *Image Processing, 2006 IEEE International Conference on*, pages 1189 –1192, 8–11 2006.

[Val05] Arne Valberg. *Light Vision Color.* The Norwegian University of Science and Technology, John Wiley & Sons, 2005.

[War04] Colin Ware. *Information Visualization: Perception for Design.* Morgan Kaufmann, Amsterdam, 2. edition, 2004.

[Zab94] Jan Zabrodzki, editor. *Grafika komputerowa, metody i narzędzia.* WNT, 1994.

[ZSAL08] Elena Zudilova-Seinstra, Tony Adriaansen, Robert van Liere. *Trends in Interactive Visualization: State-of-the-Art Survey.* Springer Publishing Company, Incorporated, 2008.

# Spis kodów źródłowych

# Skorowidz

Akademia Górniczo-Hutnicza
im. Stanisława Staszica w Krakowie
Wydział Elektrotechniki, Automatyki,
Informatyki i Elektroniki

Publikacja współfinansowana przez Unię Europejską
w ramach Europejskiego Funduszu Społecznego

Egzemplarz bezpłatny

Wydawnictwa AGH, Kraków 2011

ISBN 978-83-7464-376-4