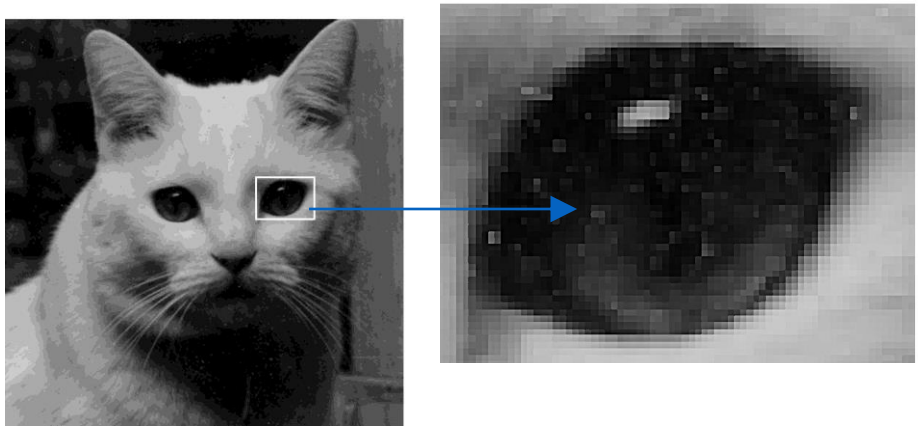
Wprowadzenie do grafiki komputerowej

Dr inż. Sławomir Samolej
D108A , tel.: 17 7432055,
email: ssamolej@kia.prz.edu.pl
www: ssamolej.kia.prz.edu.pl

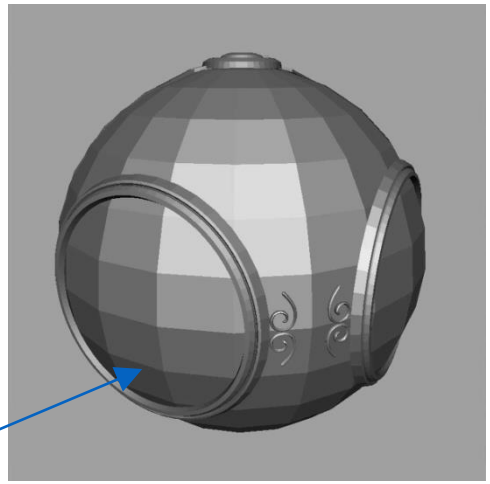
Zasada generowania obrazu

Grafika rastrowa:

- Obraz jest tworzony przez tablicę dwuwymiarową (raster)
- Składa się z pikseli zgromadzonych buforze ramki



- Bufor ramki opisywany jest przez głębokość (ilość bitów opisujących kolor piksela) oraz rozdzielczość (ilość pikseli w buforze)
- Bufor ramki o głębokości 1 bitu pozwala na rysowanie 2-kolorowego rysunku, 8-bitowy – 256 kolorowego, true color – 24 bity
- Pojawia się możliwość wypełnienia wielokątów kolorami

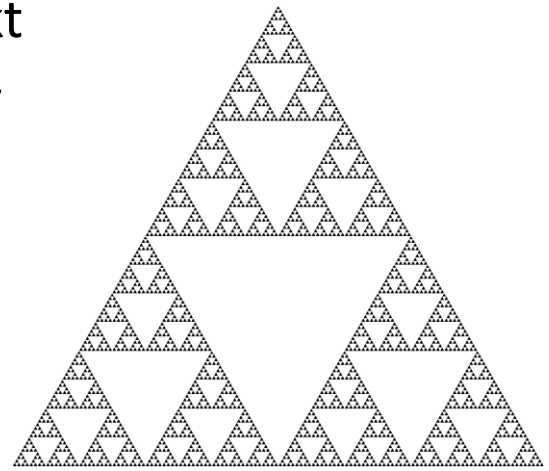


Zastosowania grafiki komputerowej

- Wyświetlanie informacji
  - Wykresy, Mapy
  - Tomografia komputerowa, rezonans magnetyczny, USG
  - Biologia molekularna, fizyka, bioinformatyka – reprezentacja dużej ilości danych w postaci wzorców graficznych.
- Projektowanie
  - CAD: Architektura, Mechanika, projektowanie procesorów
- Symulacja i animacja
  - Symulatory lotu
  - Filmy
  - Rzeczywistość wirtualna, gry
- Interfejsy użytkownika
  - X Window/Microsoft Windows/Macintosh
  - interfejsy programów CAD

Polskie akcenty

Sierpiński gasket



- Wacław Franciszek Sierpiński zaproponował obiekt matematyczny nazywany trójkątem Sierpińskiego – jeden z pierwszych fraktali

- Marek Hołyński zaproponował programową wersję biblioteki OpenGL

Czym dysponujemy?

- Biblioteki graficzne (OpenGL/DirectX)
- Programy do interaktywnej generacji siatek i animacji szkieletowych (3D Studio Max, Blender, Maya, Lightwave, Cinema4D)
- Silniki Graficzne (Darmowe: Pand3D, Ogre3D, XNA, DarkGDK, Jmonkey), (Darmowe(?): Unity3d, Unreal Engine)

Biblioteki graficzne

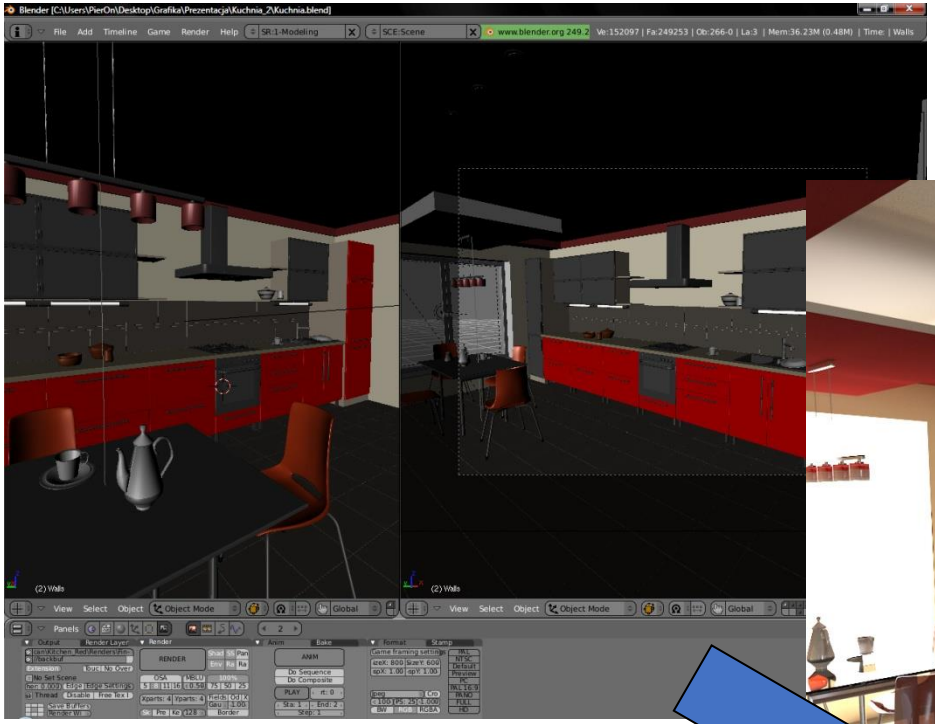
- Udostępniają tzw. prymitywy graficzne, z których można tworzyć siatki
- Pozwalają na cieniowanie, teksturowanie
- Udostępniają transformacje przestrzenne
- Umożliwiają przechowywanie współrzędnych wierzchołków i tekstur
- **Umożliwiają definiowanie shaderów**

Programy do interaktywnej generacji siatek i animacji szkieletowych (1)

- W założeniu służą do:
  - Generacji wygenerowanych sztucznie zdjęć
  - Tworzenia filmów

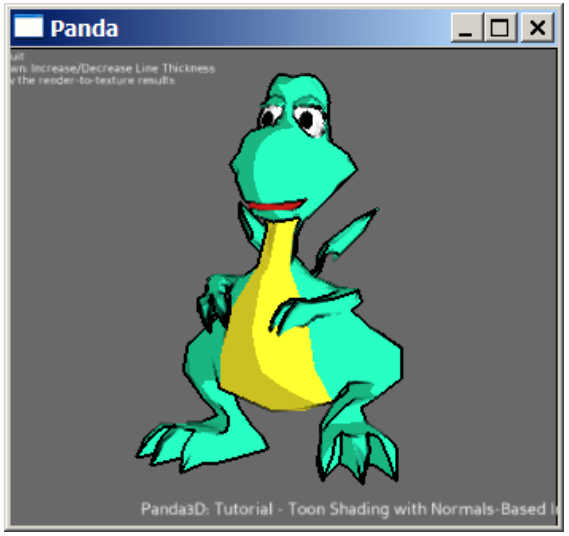
Programy do interaktywnej generacji siatek i animacji szkieletowych (2)

- Przykład modelu przed i po renderingu



Silniki graficzne / Silniki do gier

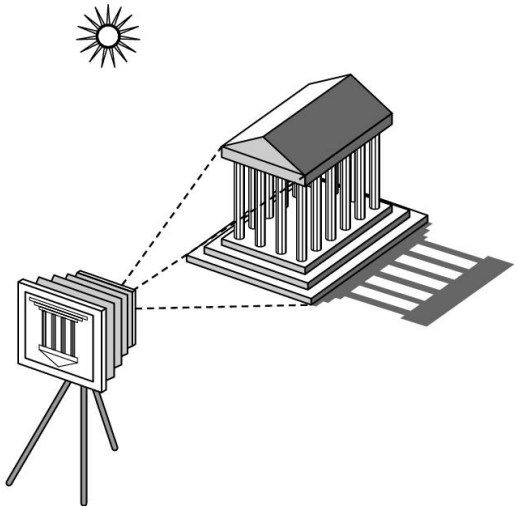
- Są w stanie wczytać pliki graficzne wygenerowane przez programy do generacji siatek i animacji szkieletowych i efektywnie nimi zarządzać
- Służą do szybszego tworzenia gier oraz interaktywnych aplikacji graficznych



Tworzenie obrazu

- W grafice komputerowej tworzone są dwuwymiarowe obrazy z zastosowaniem technik używanych w
  - Kamerach
  - Mikroskopach
  - Teleskopach
  - Ludzkiego systemu wizyjnego

Składniki tworzenia obrazu

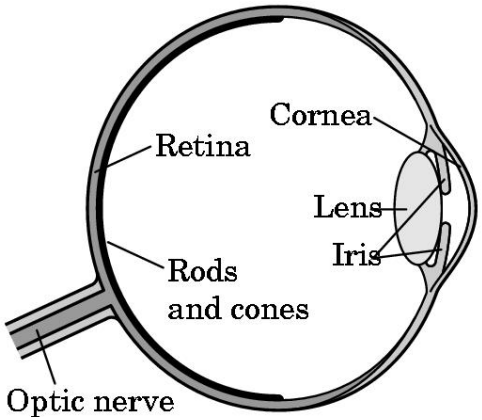


- Obiekty
- Obserwator
- Źródła światła

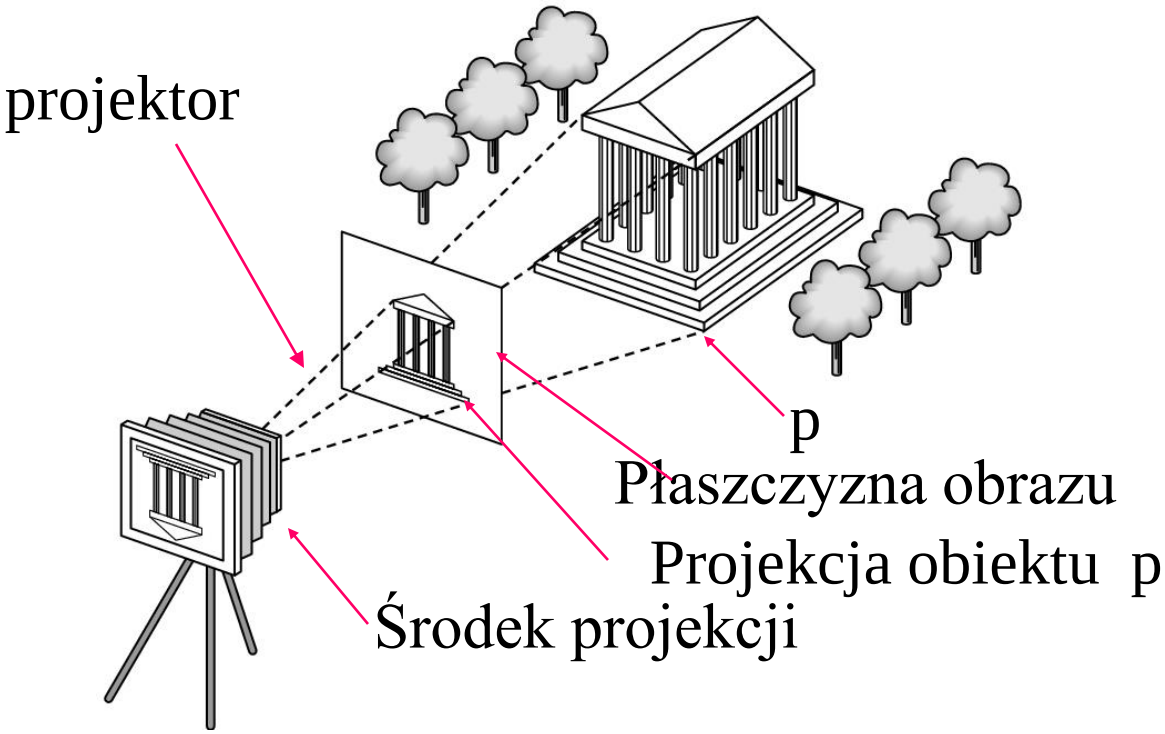
- Definiuje się atrybuty obrazu decydujące jak światło reaguje na materiały na scenie.
- Obiekty, światło i obserwator są niezależne od siebie

3 kolory

- Oko ma 3 rodzaje receptorów
  - -Pręciki (kształty, ruch)
  - -Czopki
    - kolory
    - Trzy rodzaje
    - Tylko trzy sygnały są wysyłane do mózgu
- Potrzebujemy tylko 3 kolorów do określenia obrazu



Sztuczny model kamery



Zalety

- Oddzielenie obiektu, obserwatora i źródeł światła
- Obraz dwuwymiarowy jest specjalnym przypadkiem trójwymiarowego
- Można stworzyć prosty API
  - Określ obiekty, światła, kamerę i atrybuty
  - Niech biblioteka wygeneruje obraz
- Łatwo jest taki opis zaimplementować sprzętowo

Tworzenie obrazu

- Czy możemy naśladować sztuczny model kamery do projektowania sprzętowo-programowych systemów renderowania?
- API
  - Potrzebujemy tylko wyspecyfikować
    - Obiekty
    - Materiały
    - Obserwatora
    - Światła
- Ale jak to zaimplementować?

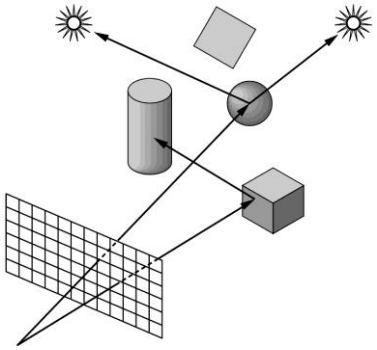
Podejście fizyczne

- Śledzenie promieni: śledzimy promienie biegnące od źródła światła dotąd gdy zostaną zaabsorbowane lub uciekną w nieskończoność Możemy łatwo zamodelować:
  - Wielokrotne odbicia
  - Półprzezroczyste obiekty

  -Wolne

  -Trzeba mieć dostęp

  do całości danych o scenie



- Radiosity:

Metoda analizy rozkładu energii

  - Bardzo wolne

Podejście praktyczne

- Przetwarzamy pojedyncze obiekty
  - Uwzględniamy lokalne oświetlenie
- Architektura potokowa

- Wszystkie etapy można zaimplementować w sprzęcie.

Vertex Processing

- Definiowanie obiektów w układach współrzędnych, konwersja pomiędzy układami współrzędnych
  - Położenie obiektów
  - Położenie kamery
  - Położenie sceny
- Zmiana koordynat = odpowiednie operacje macierzowe
- Dodatkowo przetwarza się kolory wierzchołków

Projekcja

- *Projekcja to powiązanie informacji o obiektach i kamerze i przekształcenie tej informacji na widok 2D*

  - Projekcja perspektywiczna
  - Projekcja równoległa

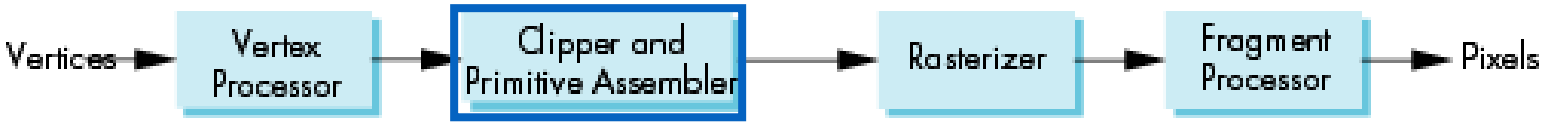
Składanie Prymitywów

- Wierzchołki muszą zostać połączone w obiekty geometryczne aby dokonać ich rasteryzacji

-linie

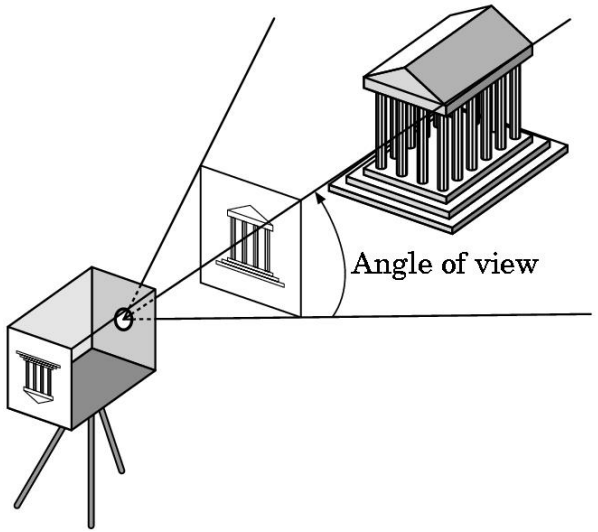
-wielokąty

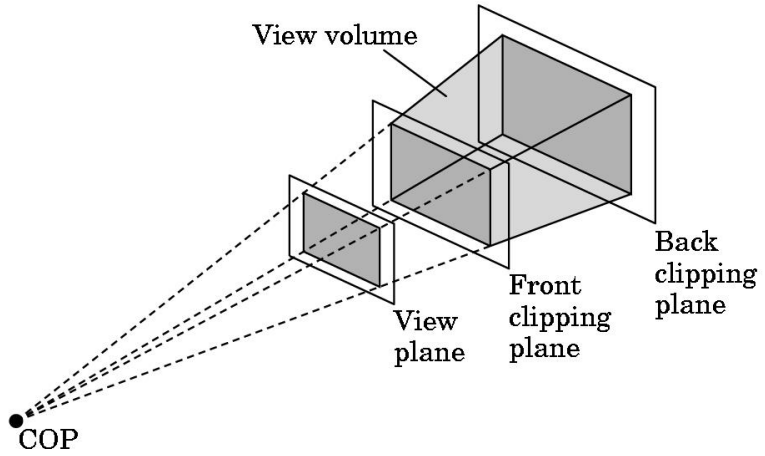
-krzywe i powierzchnie



Obcinanie (clipping)

- Podobnie jak kamera nie może widzieć całego świata, w kamera wirtualna kamera może widzieć tylko część świata. Obiekty poza ostrosłupem widzenia są „obcinane”.





Rasteryzacja

• Jeśli obiekt nie jest obcięty, to dla niego muszą zostać ustalone kolory pikseli w buforze ramki

• Rasteryzer generuje zbiór „fragmentów” dla każdego obiektu.

• Fragmenty to potencjalne piksele

- Określone jest ich położenie w buforze ramki
- Mają zdefiniowane kolory i głębokość

- Interpretowane są tu też atrybuty wierzchołków

Przetwarzanie fragmentów

- Określenie kolorów pikseli
- Uwzględnienie teksturowania
- Przysłanianie obiektów
  - Usuwanie niewidocznych krawędzi

Składniki API

- Funkcje specyfikujące co potrzebujemy do utworzenia obrazu:
  - Obiekty
  - Obserwatora
  - Źródła światła
  - Materiały
- Pozostałe informacje:
  - Wejście z urządzeń sterujących (mysz, klawiatura, game pad)
  - Możliwości systemu

Specyfikacja obiektów

- Większość API wspiera ograniczony zbiór prymitywów opisujących:
  - Punkty (obiekty 0D)
  - Segmenty linii (obiekty 1D)
  - Wielokąty (obiekty 2D)
  - Krzywe i powierzchnie
    - Kwadryki
    - Krzywe parametryczne
- Wszystkie są zdefiniowane przez ustalenie wierzchołów w przestrzeni.

Przykład

Typ obiektu

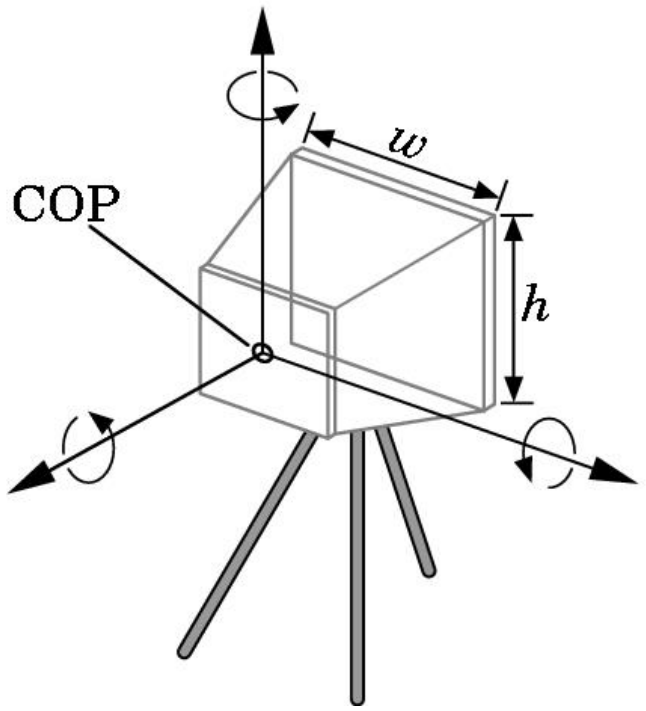
Położenie wierzchołów

```
glBegin(GL_TRIANGLES)
  glVertex3f(0.0, 0.0, 0.0);
  glVertex3f(0.0, 1.0, 0.0);
  glVertex3f(0.0, 0.0, 1.0);
glEnd( );
```

Koniec definicji obiektu

Specyfikacja kamery

- Sześć stopni swobody
  - -Pozycja obiektywu
  - -Orientacja obiejtywu
- Obiektyw
- Rozmiar filmu
- Orientacja filmu



Światła i Materiały

- Typy świateł
  - Punktowe lub rozproszone
  - Reflektory
  - Bliskie lub dalekie
  - Posiadające kolor
- Właściwości materiałów
  - Absorbcja: właściwości kolorów
  - Rozpraszanie
    - Światło rozproszone
    - Światło odbić

Grafika interaktywna kontra obliczana off-line

- Biblioteki grafiki interaktywnej:
  - OpenGL, Java3D, DirectX
  - Obliczanie grafiki musi się odbywać na bieżąco pod wpływem zewnętrznych zdarzeń (gry, symulacje w czasie rzeczywistym, bieżąca wizualizacja)
- Grafika generowana off-line:
  - 3D Studia Max, Lightwave, Maya, Blender3D
  - Rezultatem jest zwykle film lub wirtualna fotografia
  - Osiąga się foto-realizm, ale kosztem czasu obliczeń
  - Narzędzia off-line stosuje się do szybszego generowania obiektów na rzecz grafiki interaktywnej.

OpenGL - charakterystyka

- **OpenGL jest interfejsem programowym aplikacji – zestawem funkcji umożliwiających tworzenie interaktywnej grafiki 3D.**
- **Program oparty na OpenGL musi być pisany z zastosowaniem języka programowania (C, Visual Basic, Delphi itp.).**
- **Zestaw funkcji OpenGL podzielono na 3 podstawowe składniki:**
  - **Bibliotekę AUX (nowa wersja nosi nazwę GLUT)**
    - **Narzędzie do uruchamiania aplikacji OpenGL na dowolnej platformie systemowej (nie należy do standardu).**
  - **Bibliotekę GL**
    - **Zestaw funkcji zdefiniowanych w standardzie OpenGL.**
  - **Bibliotekę GLU**
    - **Zestaw funkcji wyższego poziomu umożliwiających rysowanie, oświetlenie i teksturowanie sfer dydków i walców, korzystających z biblioteki GL.**
- **Biblioteka OpenGL jest włączona jako składnik systemów operacyjnych Windows (od Win95 OSR), Unix oraz Linux.**

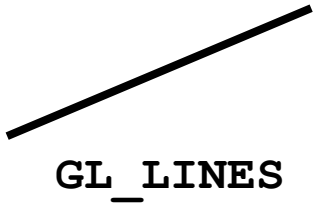
Prymitywy graficzne OpenGL - punkty

```
// Wyrysowanie dwu punktów:
glBegin(GL_POINTS);
    glVertex3f(0.0f,0.0f,0.0f);
    glVertex3f(50.0f,50.0f,50.0f);
glEnd();

// Definiowanie rozmiaru punktów:
{   GLfloat sizes[2];
    GLfloat step;

    glGetFloatv(GL_POINT_SIZE_RANGE,sizes);
    glGetFloatv(GL_POINT_SIZE_GRANULARITY,&step);

    glPointSize(GLfloat size); //ustal rozmiar punktu
}
// „Wygładzanie” punktów:
glEnable(GL_PIONT_SMOOTH);
```

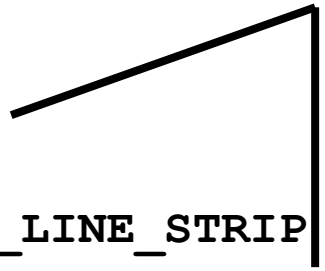
GL_POINTS

Prymitywy graficzne OpenGL - linie

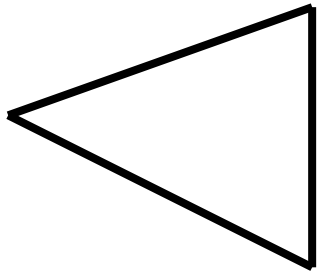
```
// Pojedyncze linie
glBegin(GL_LINES);
    glVertex3f(0.0f,0.0f,0.0f);
    glVertex3f(50.0f,50.0f,50.0f);
glEnd();
// Łańcuch linii
glBegin(GL_LINE_STRIP); // glBegin(GL_LINE_LOOP);
    glVertex3f(0.0f, 0.0f, 0.0f);     //V0
    glVertex3f(50.0f, 50.0f, 0.0f);   //V1
    glVertex3f(50.0f, 100.0f, 0.0f);  //V2
glEnd();
// Definiowanie grubości linii:
{GLfloat sizes[2];
GLfloat step;
glGetFloatv(GL_LINE_WIDTH_RANGE,sizes);
glGetFloatv(GL_LINE_WIDTH_GRANULARITY,&step);

glLineWidth(GLfloat size[1]);}
```



GL_LINES



GL_LINE_STRIP

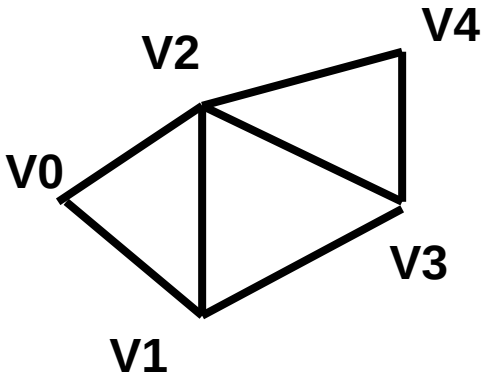


GL_LINE_LOOP

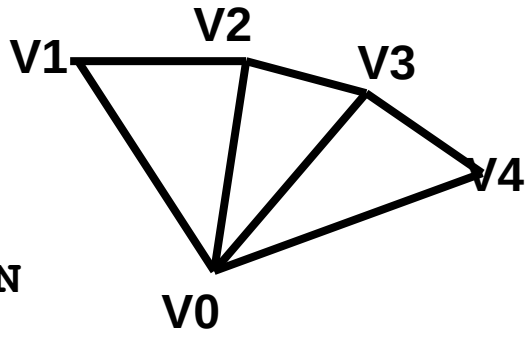
Prymitywy graficzne OpenGL – trójkąty

```
// Pojedyncze trójkąty
glBegin(GL_TRIANGLES);
    glVertex2f(0.0f, 0.0f);     //V0
    glVertex2f(25.0f, 25.0f);   //V1
    glVertex2f(50.0f, 0.0f);    //V2
    glVertex2f(-50.0f, 0.0f);   //V3
    glVertex2f(-75.0f, 50.0f); //V4
    glVertex2f(-25.0f, 0.0f);   //V4
glEnd();
// Łańcuch trójkątów
glBegin(GL_TRIANGLE_STRIP);
    //...
glEnd();
// Wachlarz trójkątów
// glBegin(GL_TRIANGLE_FAN);
    //...
glEnd();
```

GL_TRIANGLES

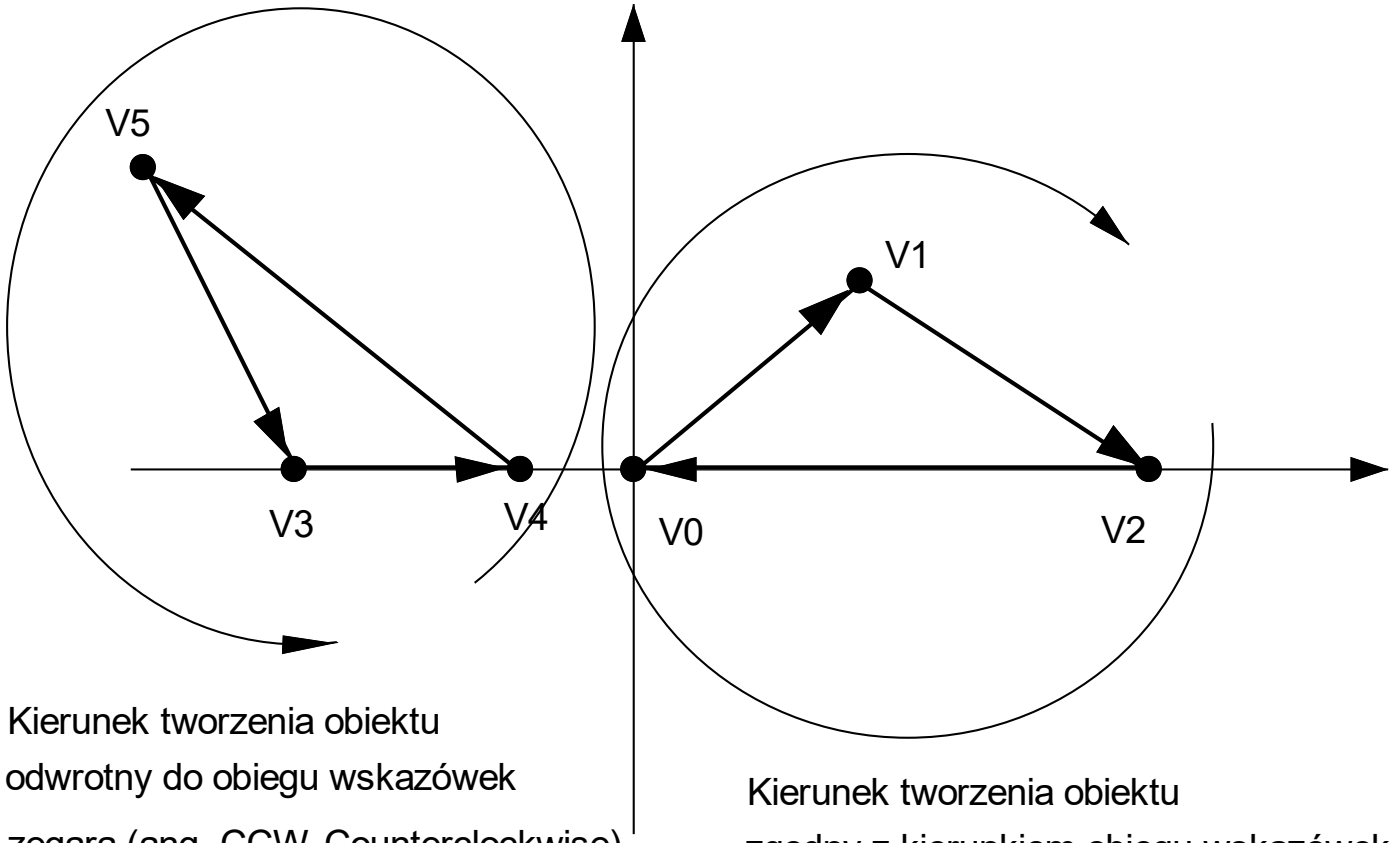


GL_TRIANGLE_STRIP



GL_TRIANGLE_FAN

Kierunek rysowania prymitywów zamkniętych



Kierunek tworzenia obiektu odwrotny do obiegu wskazówek zegara (ang. CCW-Counterclockwise)

Kierunek tworzenia obiektu zgodny z kierunkiem obiegu wskazówek zegara (ang. CW-Clockwise)

```
glFrontFace(GL_CW);  //zgodnie ze wskazówkami zegara

glFrontFace(GL_CCW); // przeciwnie do wskazówek
```

Zasady konstruowania typowej sceny

1. **Przygotowanie siatek**
   1. **Siatki skonstruować można z zastosowaniem programów graficznych lub bezpośrednio definiując prymitywy graficzne i położenie ich wierzchołków.**
   2. **Siatki zamkniętych brył warto zbudować z „zewnętrznych” ścian (można wyłączyć wtedy liczenie ścian wewnętrznych – przyspieszenie obliczeń). Do tworzenia brył należy zastosować wielokąty a nie linie, czy łamane. Przy czym w pierwszym etapie można wymusić wyświetlanie wielokątów w postaci siatek (większa czytelność).**
   3. **Każdy element sceny, który będzie się poruszał w stosunku do innych elementów należy zdefiniować jako osobną siatkę.**
   4. **W tworzeniu siatek należy się posługiwać lokalnym układem współrzędnych, nie należy definiować położenia siatki (obiektu) na scenie.**
   5. **Sąsiednie ściany brył, które współdzielą „ostrą” krawędź należy wykonać z zastosowaniem oddzielnych prymitywów.**
   6. **Poszczególne elementy siatki zamknąć w jednostka programowaych (język C – funkcje, język C++ - klasy).**
   7. **Siatki do wielokrotnego użycia (np. walec) zdefiniować w postaci sparametryzowanej (np. Funkcje z parametrami wywołania);**

Zasady konstruowania typowej sceny

2. **Zdefiniowanie transformacji przestrzennych i położenia kamery:**
   1. **Powiązanie ruchomych elementów poszczególnych obiektów za pomocą lokalnych transformacji przestrzennych;**
   2. **Ustalenie zasad ruchu obiektów i kamery na scenie.**
3. **Zdefiniowanie oświetlenia sceny:**
   1. **Włączenie oświetlenia na scenie;**
   2. **Zdefiniowanie normalnych do ścian obiektów, które będą oświetlane;**
   3. **Dla elementów nieoświetlanych dynamicznie wyłączać oświetlenie.**
4. **Teksturowanie:**
   1. **Przekształcić pliki graficzne w tekstury OpenGL;**
   2. **Powiązać współrzędne tekstur ze współrzędnymi wierzchołków.**
5. **Zdefiniowanie interfejsu użytkownika:**
   1. **Zaproponować zestaw klawiszy lub okien i elementów menu, które będą służyć do wpływania przez użytkownika na elementy sceny;**
   2. **Zastosować możliwość „dotykania” obiektów sceny:**
6. **Dodatkowe elementy – mgła, przezroczystość, mapy oświetlenia itp..**

OpenGL – modelowanie/obserwacja

- Predefiniowane funkcje OpenGL do transformacji układów współrzędnych:

```
{
double b = 1.0;
double a = 1.0;
static double angle =0.0;
static double x =0.0;

glLoadIdentity();
glTranslated(x,0,0);
//void glTranslatef(GLfloat x, GLfloat y, GLfloat z);
glRotated(angle,0,0,-1);
//void glRotatef(GLfloat angle,GLfloat x,GLfloat y, GLfloat z);
angle+=a;
x+=b;
glRectd(-20.0,-20.0,20.0,20.0);}
```

- Trzecia funkcja umożliwia zdefiniowanie skalowania:

```
void glScalef(GLfloat z,GLfloat y,GLfloat z);
```

OpenGL – zaawansowane składanie przekształceń

- Składnie kilku przekształceń:

```
{
    static double rot1=0.0, rot2=0.0;
    glLoadIdentity();
    glRectd(-10.0,-10.0,10.0,10.0);

    glRotated(rot1,0,0,1);
    glTranslated(30,0,0);
    glRotated(rot2,0,0,1);

    glRectd(-10.0,-10.0,10.0,10.0);
    rot1+=1.0;
    rot2-=2.0;
}
```

OpenGL – zaawansowane składanie przekształceń

- Funkcje glPushMatrix(); i glPopMatrix();

```
{   static double rot1=0.0, rot2=0.0;
    glLoadIdentity();
    glRectd(-10.0,-10.0,10.0,10.0);

    glPushMatrix();
    glRotated(rot1,0,0,1);
    glTranslated(30,0,0);
    glRotated(rot2,0,0,1);
    glRectd(-10.0,-10.0,10.0,10.0);
    glPopMatrix();

    glPushMatrix();
    glRotated(-rot1,0,0,1);
    glTranslated(60,0,0);
    glRotated(-rot2,0,0,1);
    glRectd(-10.0,-10.0,10.0,10.0);
    glPopMatrix();

    rot1+=1.0;
    rot2-=2.0;
}
```

OpenGL – model oświetlenia

- Składowe światła OpenGL
  - Światło otaczające (ambient)

    Nie pochodzi z żadnego określonego kierunku. Powoduje równomierne oświetlenie obiektów na wszystkich powierzchniach i wszystkich kierunkach.
  - Światło rozproszone (diffuse)

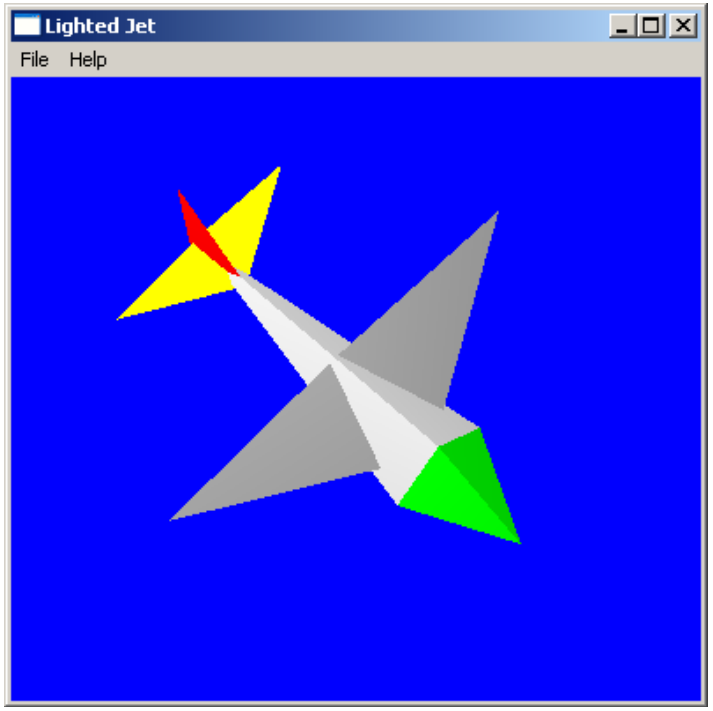
    Pochodzi z określonego kierunku, odbijane jest od powierzchni równomiernie. Powierzchnie intensywniej oświetlone są jaśniejsze od mniej oświetlonych.
  - Światło odbłysków (kierunkowe) (specular)

    Biegnie z określonego kierunku, odbijane jest w ściśle określonym kierunku.
- Przykład – definiowanie źródła światła laserowego:

	Czerwony (R)	Zielony (G)	Niebieski (B)	Alpha
Św. kierunkowe	0,99	0,0	0,0	1,0
Św. rozproszone	0,10	0,0	0,0	1,0
Św. otaczające	0,05	0,0	0,0	1,0

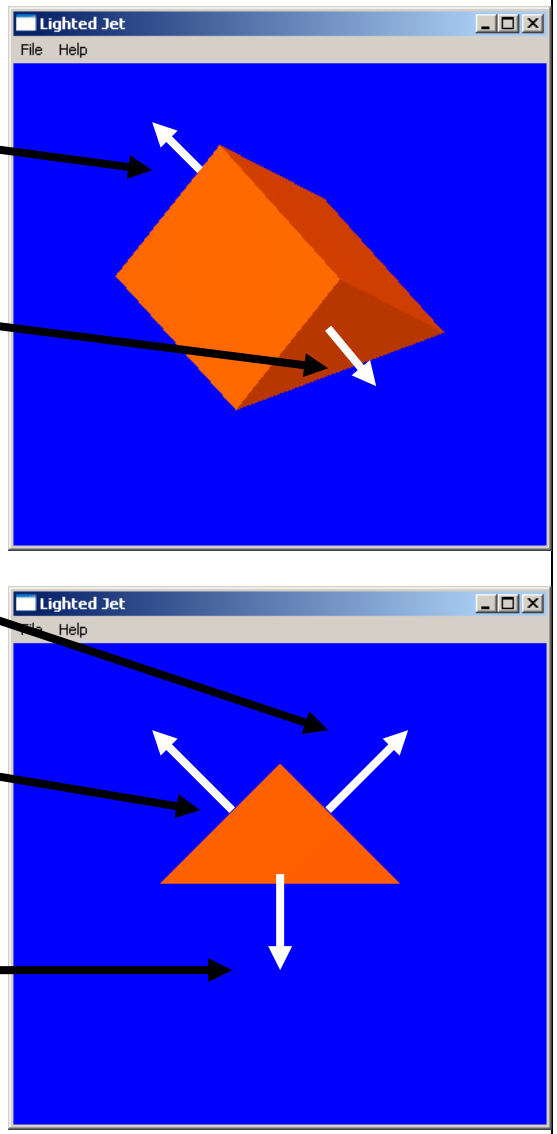
OpenGL – scena oparta na świetle rozproszonym

```
{GLfloat  ambientLight[] = { 0.5f, 0.5f, 0.5f, 1.0f };
 GLfloat  diffuseLight[] = { 0.7f, 0.7f, 0.7f, 1.0f };
 GLfloat   lightPos[] = { -50.f, 50.0f, 100.0f, 1.0f };

 glEnable(GL_LIGHTING);

 glLightfv(   GL_LIGHT0,
              GL_AMBIENT,
              ambientLight);
 glLightfv(   GL_LIGHT0,
              GL_DIFFUSE,
              diffuseLight);
 glLightfv(   GL_LIGHT0,
              GL_POSITION,
              lightPos);
 glEnable(GL_LIGHT0);

 glEnable(GL_COLOR_MATERIAL);
 glColorMaterial(GL_FRONT, GL_AMBIENT_AND_DIFFUSE);
 ...
}
```



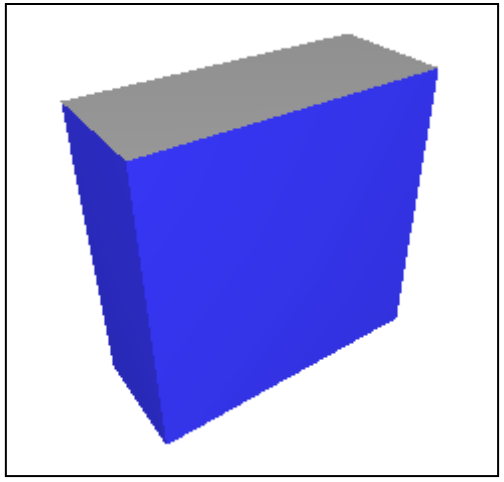
OpenGL – podstawowe definiowanie normalnych

```
glBegin(GL_TRIANGLES);
glNormal3d(0,0,-1);
glVertex3d(10,0,-10); glVertex3d(-10,0,-10);
glVertex3d(0,10,-10);
glNormal3d(0,0,1);
glVertex3d(10,0,10); glVertex3d(0,10,10);
glVertex3d(-10,0,10); glEnd();

glBegin(GL_QUADS);
glNormal3d(sqrt(2)/2,sqrt(2)/2,0);
glVertex3d(10,0,10);  glVertex3d(10,0,-10);
glVertex3d(0,10,-10); glVertex3d(0,10,10);

glNormal3d(-sqrt(2)/2,sqrt(2)/2,0);
glVertex3d(-10,0,10); glVertex3d(0,10,10);
glVertex3d(0,10,-10); glVertex3d(-10,0,-10);

glNormal3d(0,-1,0);
glVertex3d(10,0,10);  glVertex3d(-10,0,10);
glVertex3d(-10,0,-10);glVertex3d(10,0,-10);
glEnd();
```



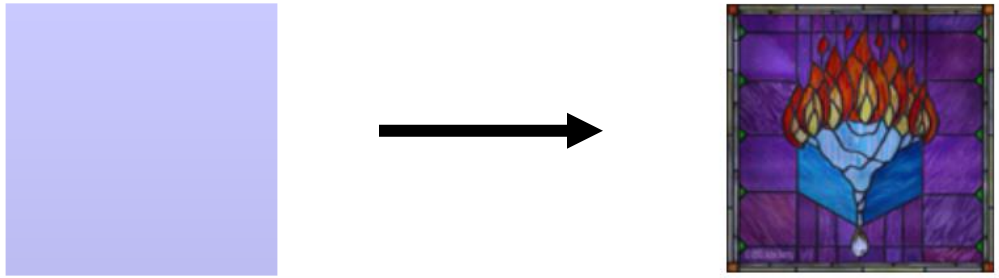
OpenGL – teksturowanie

- **Teksturowanie polega na pokrywaniu wielokątów obrazami (plikami graficznymi)**
- **Umożliwia znaczące zwiększenie realizmu sceny przy niewielkim zwiększeniu nakładu obliczeniowego**
- **Rozwój akceleratorów graficznych w ciągu ostatnich lat skupiał się na rozbudowie sprzętowych funkcji wspomagających przechowywanie, szybki transfer i rozkładanie tekstur na elementach sceny**
- **Większość efektów symulujących oświetlenie w grach komputerowych realizowana jest z zastosowaniem teksturowania**



Teksturowanie – pokrywanie obiektów teksturą

```
void kwadrat(void)
{   glColor3d(0.7,0.7,0.9);
    // Określ bieżący obiekt tekstury:
    glBindTexture(GL_TEXTURE_2D,texture[0]);
    glEnable(GL_TEXTURE_2D); // Włącz teksturowanie
    glBegin(GL_QUADS);
        glNormal3d(0,0,1);
        // Powiąż współrzędne tekstury z wierzchołkami:
        glTexCoord2d(1.0,1.0); glVertex3d(25,25,25);
        glTexCoord2d(0.0,1.0); glVertex3d(-25,25,25);
        glTexCoord2d(0.0,0.0); glVertex3d(-25,-25,25);
        glTexCoord2d(1.0,0.0); glVertex3d(25,-25,25);
    glEnd();
    glDisable(GL_TEXTURE_2D); // Wyłącz teksturowanie
}
```



Techniki zwiększania realności w grach komputerowych

- Użytkownicy oczekują poziomu odwzorowania wirtualnego świata na poziomie zbliżonym do rzeczywistości (Final Fantasy, Ekspres polarny, Opowieść Wigilijna 2009,
  Avatar)
- Możliwości interaktywnej grafiki 3D (zwłaszcza na platformę PC) są wciąż ograniczone.

Gra

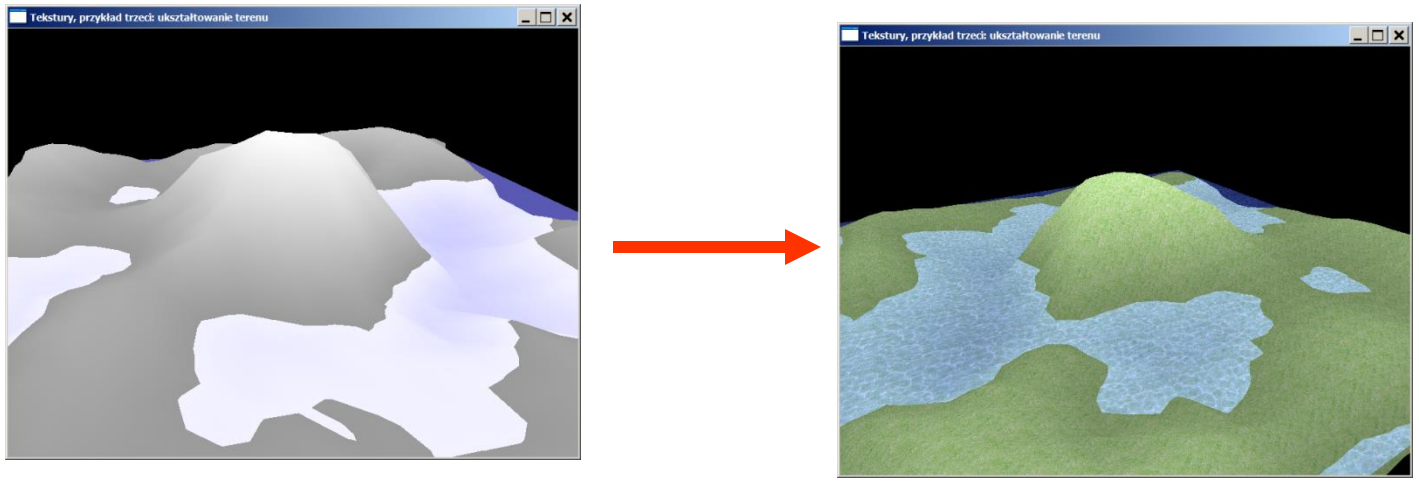
Film

Uwaga: KONSOLE!!!

- Poziom grafiki w nowych grach konsolowych przewyższa gry na PC !

Teksturowanie - dyskusja

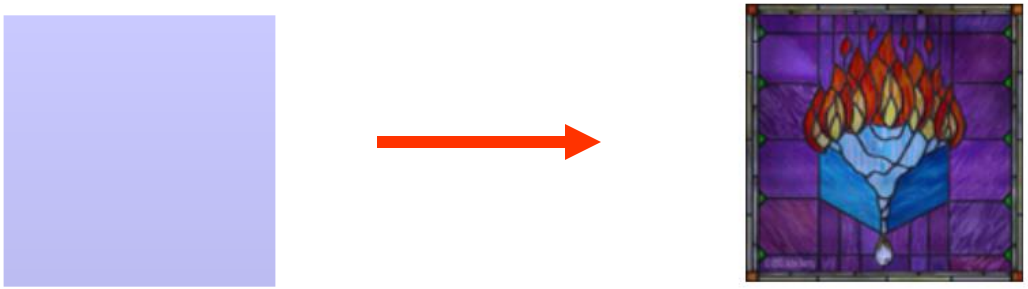
- Teksturowanie polega na pokryciu obrazem (1D/2D/3D) wielokąta
- Skomplikowane modele mogą być zastąpione prostszymi pokrytymi teksturami
- Inteligentne teksturowanie może zastąpić liczenie oświetlenia
- Jednym z głównych nurtów rozwoju kart graficznych jest powiększanie możliwości lokalnego gromadzenia tekstur i szybkiego teksturowania obiektów na scenie
- Nawet proste teksturowanie zwiększa realność sceny 3D.



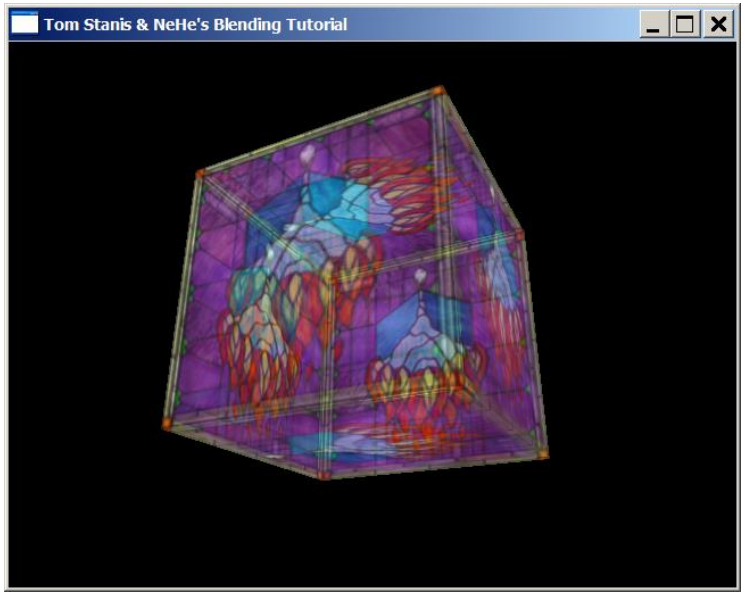
Podstawowe teksturowanie

- Załaduj plik graficzny
- Skonwertuj plik graficzny na teksturę
- Ustal filtrowanie
- Włącz teksturowanie
- Połącz współrzędne tekstury z wierzchołkami obiektu graficznego

```
glBegin(GL_QUADS);
    glTexCoord2d(1.0,1.0); glVertex3d(25,25,25);
    glTexCoord2d(0.0,1.0); glVertex3d(-25,25,25);
    glTexCoord2d(0.0,0.0); glVertex3d(-25,-25,25);
    glTexCoord2d(1.0,0.0); glVertex3d(25,-25,25);
glEnd();
```



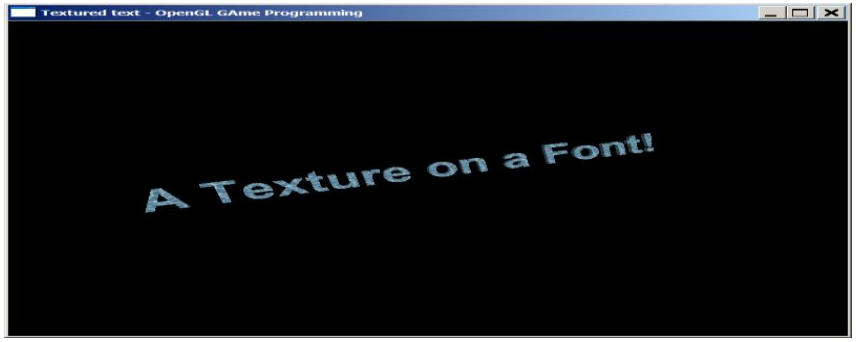
Teksturowanie z przezroczystością



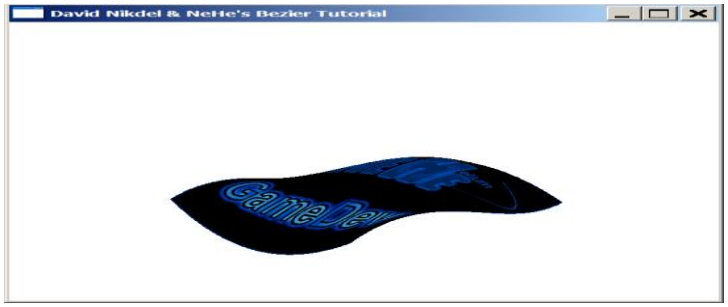

```
//…
glDepthFunc(GL_LEQUAL);
glBlendFunc(GL_SRC_ALPHA,GL_ONE);
//…
if(blend)
{ glEnable(GL_BLEND);
  glDisable(GL_DEPTH_TEST);
}
```

Możliwości teksturowania w OpenGL

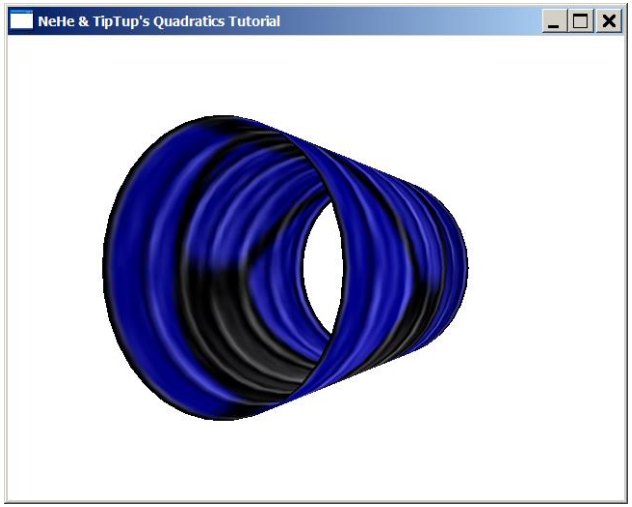
Fonts:



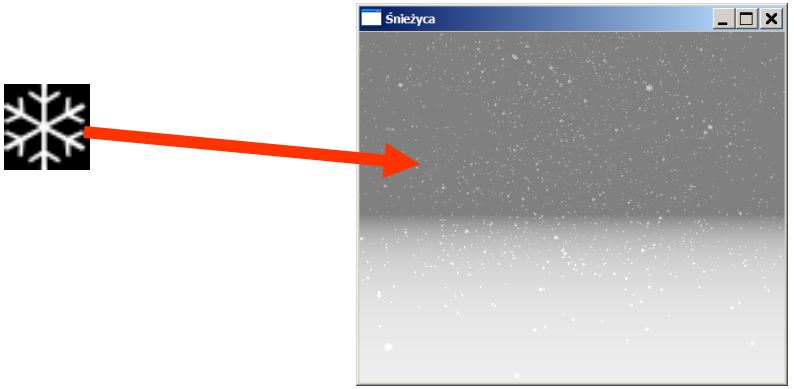
OpenGL surfaces (powierzchnie):



OpenGL quadrics (Kwadryki):



Particles (cząsteczki):

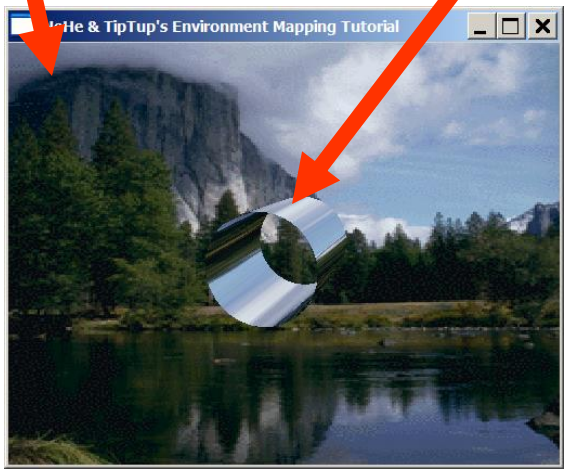


Environment Mapping

- Udawane odbicia (gry samochodowe):

Normalny obrazek:

Obrazek zniekształcony sferycznie



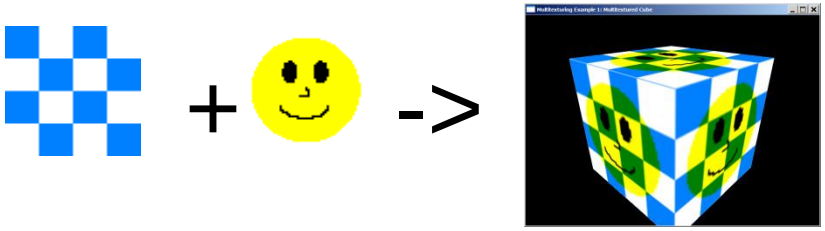

```
// THE MOST IMPORTANT COMMANDS:
//…
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE,
GL_SPHERE_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE,
GL_SPHERE_MAP);
//…
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
//
// gluCylinder(…);
//
glDisable(GL_TEXTURE_GEN_S);
glDisable(GL_TEXTURE_GEN_T);
```

Multiteksturowanie

- Możliwość komponowania jednej tekstury z kilku
- Dokonując tylko szybkich wymian tekstur osiąga się interesujące efekty:
  - Fałszywego oświetlenia (light maps)
  - Fałszywego odwzorowywania chropowatości (false bump mapping)

Multiteksturowanie podstawy

- Potrzebujemy 2 tekstur:

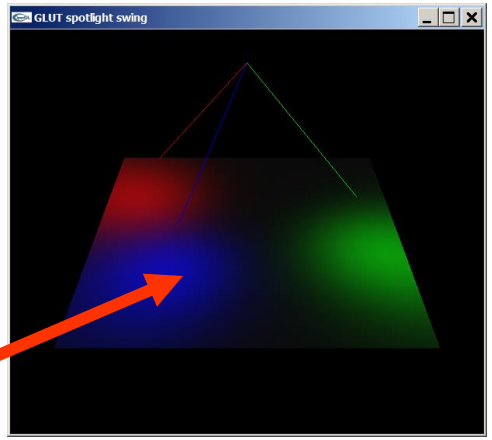


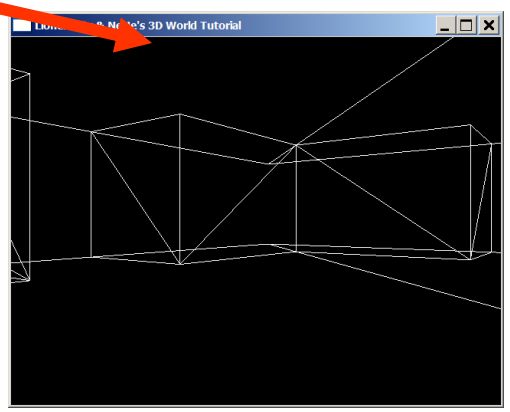
- **W OpenGL musi działać rozszerzenie GL_ARB_multitexture**:

```
glActiveTextureARB(GL_TEXTURE0_ARB);  glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, smileTex->texID);
glActiveTextureARB(GL_TEXTURE1_ARB);  glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, checkerTex->texID);
//…
glBegin(GL_QUADS); //…
    glMultiTexCoord2fARB(GL_TEXTURE0_ARB, 1.0f, 0.0f);
    glMultiTexCoord2fARB(GL_TEXTURE1_ARB, 1.0f, 0.0f);
    glVertex3f(0.5f, 0.5f, 0.5f);
//…
```

Model oświetlenia OpenGL - dyskusja

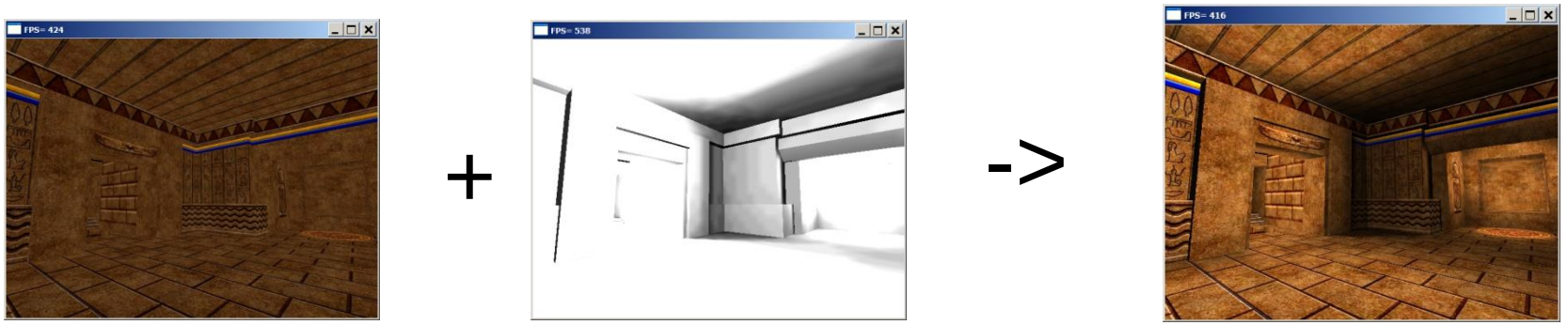
- W modelu Ponga:
  - Każdy wielokąt oświetlany jest niezależnie
  - Światło rozproszone na jednym wielokącie nie wpływa na inny
  - Cieniowanie odbywa się dla każdego wierzchołka
- Żeby uzyskać ładnie ocieniowaną powierzchnię trzeba ją podzielić na kilkadziesiąt mniejszych wielokątów.
- W grach komputerowych:
  - Dalej poszukuje się redukcji złożoności siatek
  - Nie oświetla się wszystkich elementów sceny
  - Użytkownicy oczekują realistycznych efektów
- SPRYTNIE ZASTOSOWANE MULTITEKSTUROWANIE MOŻE ZASTĄPIĆ NAKŁAD OBLICZENIOWY NA CIENIOWANIE Z ZASTOSOWANIEM OŚWIETLENIA OPENGL





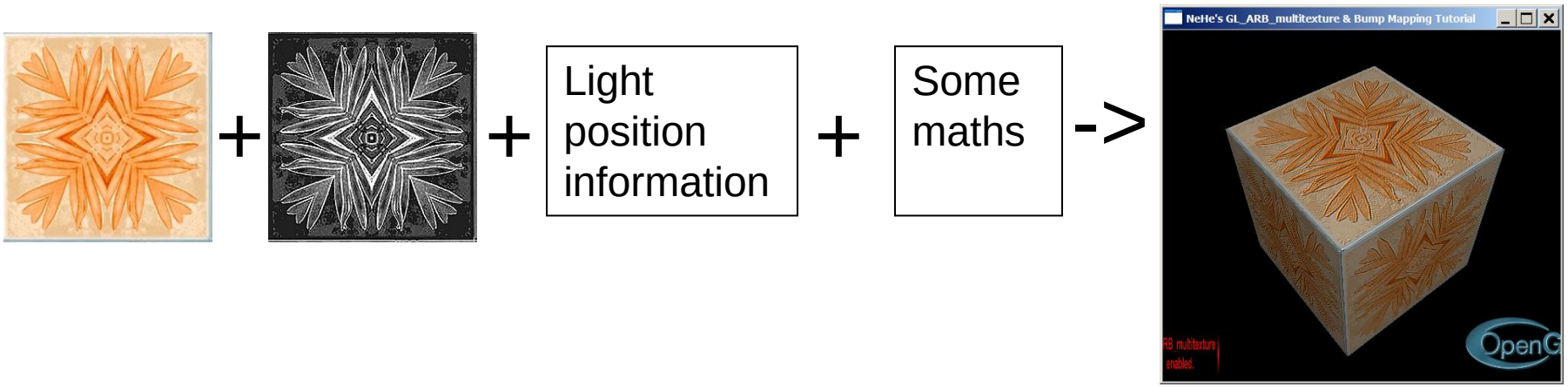
Mapy oświetlenia

- Wystarczy nałożyć na ściany kombinację tekstur odzwierciedlających fakturę powierzchni i oświetlenie, żeby uzyskać dobry efekt fałszywego oświetlenia



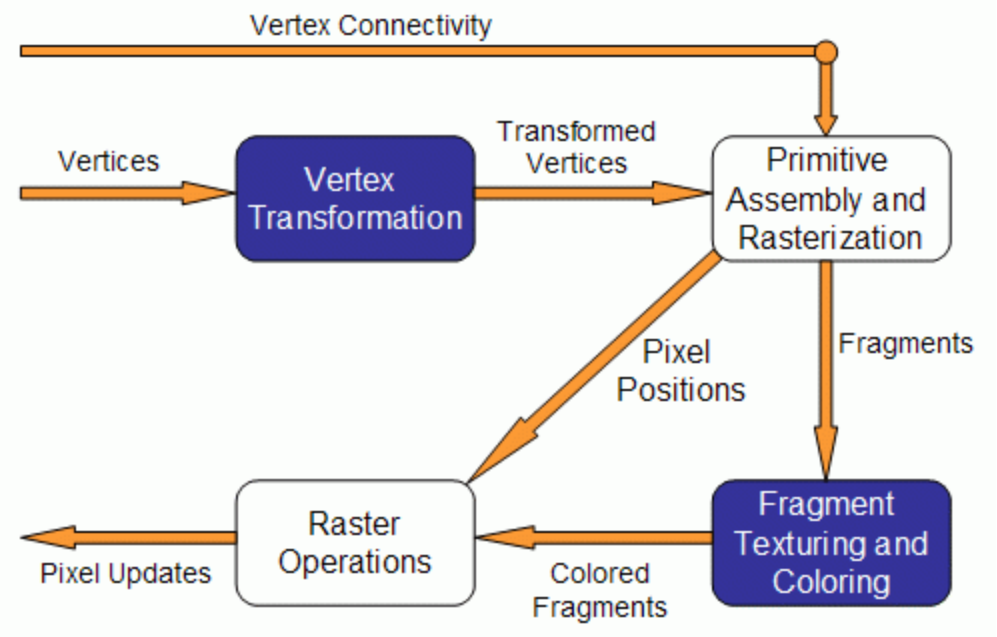
Fałszywa chropowatość

- Bada się położenie źródła światła, obserwatora oraz orientację oświetlanego obiektu
- Na podstawie wymienionych danych dokonuje się niewielkiego przemieszczenia odpowiednio przygotowanych 2 tekstur na płaszczyźnie.

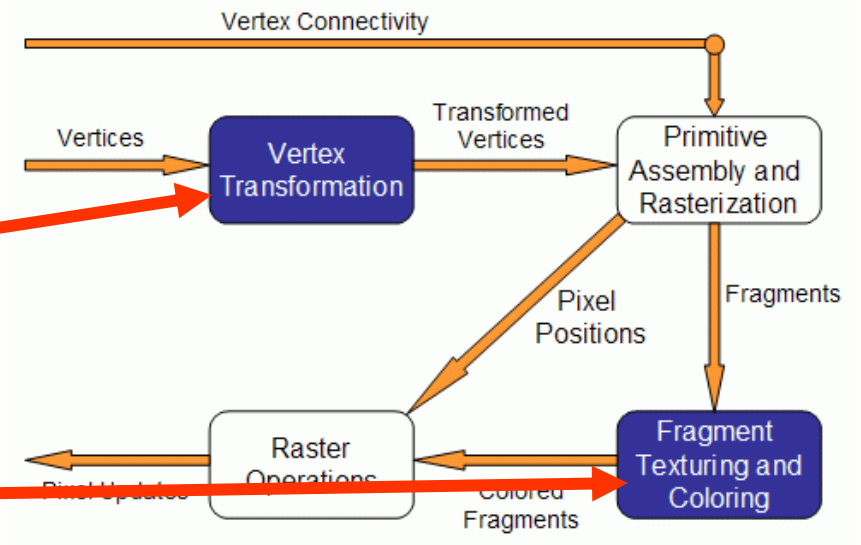


Standardowa linia potokowa OpenGL

- Transformacje wierzchołków:
  - Położenie, kolor, normalna, współrzędne tekstury
- Utworzenie prymitywów i Rasteryzacja:
  - Powiązanie wierzchołków z zastosowaniem prymitywów
  - Transformacje wycinające
  - Zdefiniowanie fragmentów – potencjalnych pikseli
- Teksturowanie fragmenntów i kolorowanie :
  - Kolory fragmentów są mieszane z kolorami tekstur
  - Obliczana jest mgła
- Operacje rastrowe:
  - Test Apha, Stencil, Głębokość
  - Ustalenie wartości koloru piksela

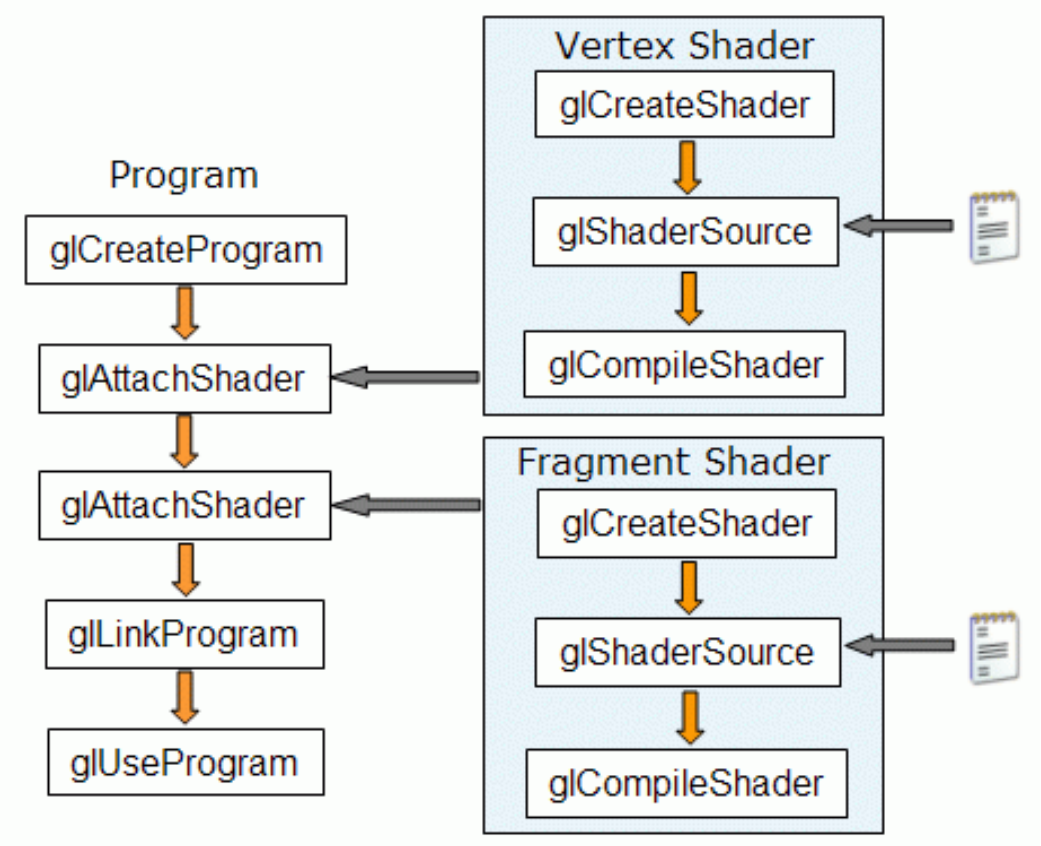


Możliwość wprowadzenia shaderów



- **Vertex shader** – możliwość modyfikowania stanu wierzchołków
- **Fragment shaders –** możliwość modyfikowania stanu fragmentów

OpenGL Shading Language (GLSL)



Technika Ray-tracing

- Droga do uzyskania lepszych wyników renderowania:
  - Obraz tworzy się przez prześledzenie promieni od źródła światła do kamery.
  - Każdy promień może wchodzić w wiele interakcji z obiektami, które stoją na jego drodze do kamery
  - Technika ray-tracing bardziej odpowiada prawom fizyki
  - Można obliczyć globalny efekt oświetlenia ale procedury są wolne i stwarzają problem w aplikacjach interaktywnych
- Istnieje ścieżka rozwoju akceleratorów graficznych, w których wbudowywane są procedury ray-tracing.