

Grafika - opracowanie do kolokwium 29.05.2013

[Kolorki - choose one and contribute](#)

[Materiały](#)

[Wykłady 2012](#)

[Wykłady 2013](#)

[Notatki](#)

[Pytania z poprzednich lat](#)

[6.06.2012](#)

- [1. Coś tam o ambient, diffuse, specular.](#)
- [2. Jaki kolor kuli uzyskamy, jeżeli będziemy oświetlać niebieską, czerwoną i żółtą kulę światłem zielonym?](#)
- [5. Co to jest CMY i CMYK i gdzie się stosuje?](#)
- [6. Co to jest HSV?](#)
- [7. Co to jest interpolacja Gourauda?](#)
- [8. Czym różni się interpolacja Gourauda od interpolacji Phong?](#)
- [9. Co oznaczają punkty \(kontrolne\)? w opisie krzywych Beziera trzeciego stopnia \(interpretacja\)](#)
- [Dystans pomiędzy P0 oraz P1 oznacza jak długo krzywa porusza się w kierunku punktu P2 przed skręceniem na P3.](#)
- [10. Jakie są zalety i wady funkcji uwikłanych?](#)
- [11. Jaką transformację opisuje poniższe działanie na macierzach?](#)
- [12. Co jest najbardziej wymagające obliczeniowo \(czy tam najtrudniejsze, nie pamiętam\) w ray tracingu? Jak to wpływa na wybór obiektów na scenie?](#)
- [13. Jakie główne czynniki wpływają na czas renderingu metodą ray tracingu?](#)
- [14. Co to jest tekstura parametryczna](#)

[Pytania z tegorocznego kolosa w terminie "0":](#)

Niżej zagadnienia od Aldy

Kolorki - choose one and contribute

- Bartek (mequrel)
- bryk
- falconepl
- wojtasskorcz
- lCeQ
- Alicja
- Kamil Mielnik
- Daniel
- bp
- joe
- reve

- adebski

Materiały

Wykłady 2012

- <http://bit.agh.edu.pl/forum/download/file.php?id=4108>

Wykłady 2013

- http://galaxy.agh.edu.pl/~alda/Grafika_2013/01_opengl_wprowadzenie_2013.pptx
- http://galaxy.agh.edu.pl/~alda/Grafika_2013/02_glsi_wprowadzenie_2013.pptx

Notatki

- http://student.agh.edu.pl/~psokol/fg3xa2c/grafika_wyklad.pdf

Pytania z poprzednich lat

6.06.2012

- źródło: <http://bit.agh.edu.pl/forum/viewtopic.php?f=199&t=17460>
- “(...) były grupy (...) Wszyscy mieli 6 otwartych pytań, pożądane odpowiedzi miały być długości +/- jednego zdania.”

1. Coś tam o ambient, diffuse, specular.

Specular

W oświetleniu specular światło jest odbijane od obiektu tylko w jednym kierunku - dokładnie pod takim samym kątem jak kąt padania. Jeżeli tak odbite światło trafi do kamery, jest ono widoczne jako bardzo jasny punkt na obiekcie (zazwyczaj biała plamka).

W oświetleniu tym bierzemy więc pod uwagę kąt między wektorem światła odbitego od powierzchni (żeby go wyliczyć trzeba uwzględnić wektor od źródła światła i normalną), a wektorem ‘do obserwatora’. Jeżeli te wektory są blisko siebie (obserwator patrzy prawie dokładnie w odbite światło), to chcemy, żeby obraz był bardzo jasny. Chcemy też, żeby jasność

dodana spadała szybko z odległością, dlatego stosujemy $\cos^{duza\ potega}(\beta)$, gdzie beta to kąt między tymi wektorami (iloczyn skalarny wektorów znormalizowanych)

Ruch obserwatora ma wpływ na ten rodzaj oświetlenia.

Diffuse

W oświetleniu diffuse światło jest odbijane od obiektu w wielu kierunkach - najmocniej pod takim samym kątem jak kąt padania. Im większa różnica między kątem padania, a kątem odbicia, tym słabsze jest światło odbite.

W tym rodzaju oświetlenia bierzemy pod uwagę wektor ‘do źródła światła’ i normalną naszej powierzchni (a konkretnie wyinterpolowaną dla danego fragmentu). Chcemy, żeby światło oświetlało najmocniej powierzchnie, na które świeci prostopadle. Tak dzieje się, gdy wektor ‘do źródła światła’ jest ułożony w tym samym kierunku, co normalna. Używamy więc wzoru $\cos(\beta)$, gdzie beta, to kąt pomiędzy tymi wektorami (iloczyn skalarny wektorów znormalizowanych).

Warto wziąć pod uwagę, że jeżeli będziemy rozważać powierzchnię 'z tyłu' obiektu (nieoświetloną), to nasza normalna będzie skierowana 'w drugą stronę', przez co iloczyn skalarny wyjdzie nam <0 . Warto to wyifować.

Ruch obserwatora nie ma wpływu na ten rodzaj oświetlenia.

Ambient

Oświetlenie ambient pozwala oświetlić (ale nie tak mocno jak dwa poprzednie modele) tę stronę obiektu, na którą nie pada bezpośrednio światło ze źródła światła.

Tu warto zaznaczyć, że światło ambient tylko sprawia wrażenie, że jest światłem odbitym od innych obiektów. W rzeczywistości jest to światło bezkierunkowe, otaczające, środowiska.

Wszystkie powierzchnie oświetlone są nim jednakowo, a natężenie odbitego światła zależy jedynie od własności powierzchni.

W normalnym świecie światło odbija nam się bardzo wiele razy, dzięki czemu praktycznie nigdy powierzchnie widziane przez nas nie są zupełnie czarne. Ponieważ stosujemy uproszczony model oświetlenia (nie uwzględniający tych wszystkich odbić), w naszej scenie pojawiałyby się obszary zupełnie czarne, co byłoby nienaturalne. Dodając minimalną składową ambient sprawiamy, że wszystko jest przynajmniej 'trochę' widoczne.

Źródła:

http://www.youtube.com/watch?v=RjA_sC4bCAM

2. Jaki kolor kuli uzyskamy, jeżeli będziemy oświetlać niebieską, czerwoną i żółtą kulę światłem zielonym?



<http://painting.about.com/library/blpaint/blcolormixingpalette1.htm>

Wg (<http://samszyn.2ap.pl/1.pdf>) pierwsze dwie będą czarne (bo nie są w stanie odbić zielonego światła) a żółta będzie widziana na zielono (żółty = czerwony + zielony, więc zielone światło jest w stanie się odbić)

5. Co to jest CMY i CMYK i gdzie się stosuje?

Jest to subtraktywny model kolorów, oparty na trzech (CMY) bądź czterech barwach (CMYK):

- **C** - cyan
- **M** - magenta
- **Y** - yellow
- **K** - black

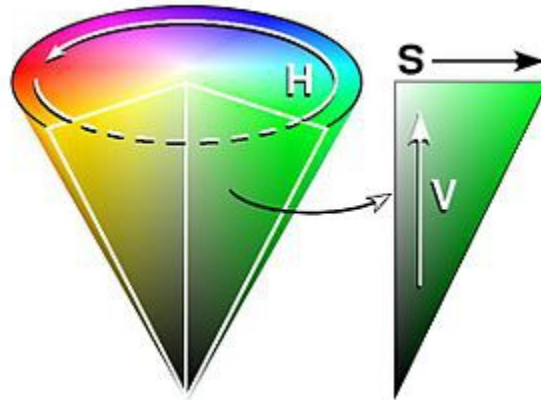
Ponieważ jest to model subtraktywny, poprzez połączenie wszystkich barw uzyskujemy kolor czarny, w odróżnieniu do modelu addytywnego (jak np. RGB), w którym połączenie wszystkich barw tworzy kolor biały. Przykładem modelu subtraktywnego może być efekt mieszania pigmentów (mieszanie farb o wielu barwach), a przykładem modelu addytywnego piksele współczesnych wyświetlaczy LCD, CRT itp.

CMYK jest głównie stosowany w druku wielobarwnym (m.in. prasa kolorowa).

6. Co to jest HSV?

HSV (Hue Saturation Value) – model opisu przestrzeni barw. Można sobie go przedstawić jako stożek, którego wymiary opisują:

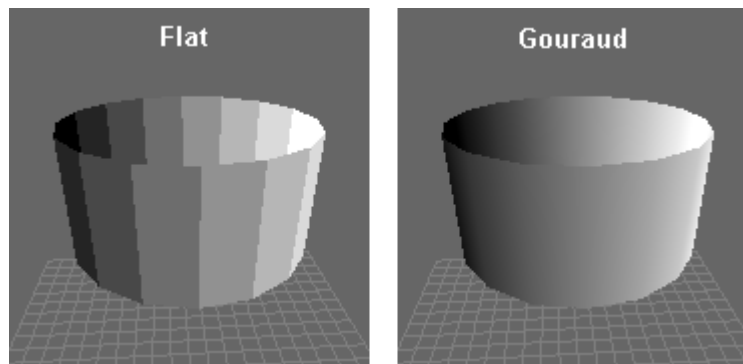
- S (Saturation, nasycenie koloru) – promień podstawy,
 - V (Value/Brightness, moc światła białego) – wysokość stożka.
- H (Hue, odcień światła) to natomiast kąt na kole barw, który przyjmuje wartości od 0 do 360 stopni (centrum barwy zielonej - 120 st., centrum barwy niebieskiej 240, czerwonej 0 lub 360).



Źródło: [http://pl.wikipedia.org/wiki/HSV_\(grafika\)](http://pl.wikipedia.org/wiki/HSV_(grafika))

7. Co to jest interpolacja Gourauda?

Jest to metoda interpolacji cieniowania pozwalająca uzyskać ciągle (continuous) cieniowanie na powierzchni złożonej z siatki trójkątów. W uproszczeniu polega na obliczeniu oświetlenia w wierzchołkach każdego z trójkątów i liniowym zinterpolowaniu tych wartości na pozostałe piksele (wnętrza trójkątów). Jest bardzo szybka i skutecznie wygładza krawędzie (również tam, gdzie może być to niepożądane), jednak nie sprawdza się w obliczaniu odbłasków. Przykład:



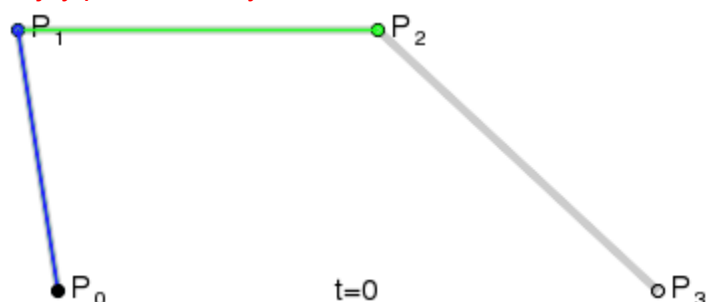
8. Czym różni się interpolacja Gourauda od interpolacji Phong?

Phong	Gouraud
Interpolujemy przez normalne sąsiadujących wielokątów	Interpolujemy przez wierzchołki trójkątów (wartość we wierzchołku = natężenie światła)
Lepsza jakość, ale wolniejsza	Gorsza jakość, ale szybsza

Generuje dobre odbłaski dzięki zastosowaniu dostosowanego do niej Phong reflection model	Nie oblicza odbłasków, a jeżeli wprowadzimy je sztucznie, to będą rozmazane
Podczas interpolacji zachowywane są kolory znajdujące się we wnętrzu trójkątów	Jeżeli kolor wewnątrz trójkąta znacznie różni się od kolorów na jego wierzchołkach (np. jest tam specular reflection), to zostanie on utracony podczas interpolacji

9. Co oznaczają punkty (kontrolne)? w opisie krzywych Beziera trzeciego stopnia (interpretacja)

W krzywych Beziera 3-go stopnia mamy cztery punkty, które wyznaczają trzy odcinki. Po tych odcinkach, podczas rysowania krzywej, poruszają się "znaczniki" które tworzą kolejne odcinki i pozwalają obliczyć kolejny punkt do narysowania.



Dystans pomiędzy P_0 oraz P_1 oznacza jak długo krzywa porusza się w kierunku punktu P_2 przed skręceniem na P_3 .

Źródło: Wikipedia

10. Jakie są zalety i wady funkcji uwikłanych?

Cytat z wykładów (11_implicit.pdf:19):

<p><i>Zalety reprezentacji uwikłanej</i></p> <ul style="list-style-type: none"> • Wizualizacja dowolnych wyrażeń algebraicznych • Zwarta notacja złożonych powierzchni • Budowanie złożonych powierzchni poprzez łączenie zbioru prostszych przy zachowaniu ciągłości powierzchni • Łatwość deformowania powierzchni 	<p><i>Wady reprezentacji uwikłanej</i></p> <ul style="list-style-type: none"> • Złożoność procesu wizualizacji • Zaleta może być również wadą (blending) • Trudność modelowania przy pomocy tzw. powierzchni algebraicznych • składanie elementów, • „gubienie masy”.
--	--

Funkcja uwikłana – funkcja, która nie jest przedstawiona jawnym przepisem, wzorem wyrażającym zależność wartości funkcji od jej argumentu, lecz bardziej złożonym związkiem, który nie daje się prosto przekształcić na jawny wzór.

Czyli w naszym (graficznym) przypadku jest to opis figury postaci $f(x, y, z) + S_0 = 0$.

11. Jaką transformację opisuje poniższe działanie na macierzach?

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 7 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 20 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix}$$

(od autora: Nie jestem pewien, czy na pewno takie macierze, ale na 90% tak.)

Źródło: 02_opengl_transformacje_2012.pdf:23,24, notatki strona 6 “składanie operatorów”

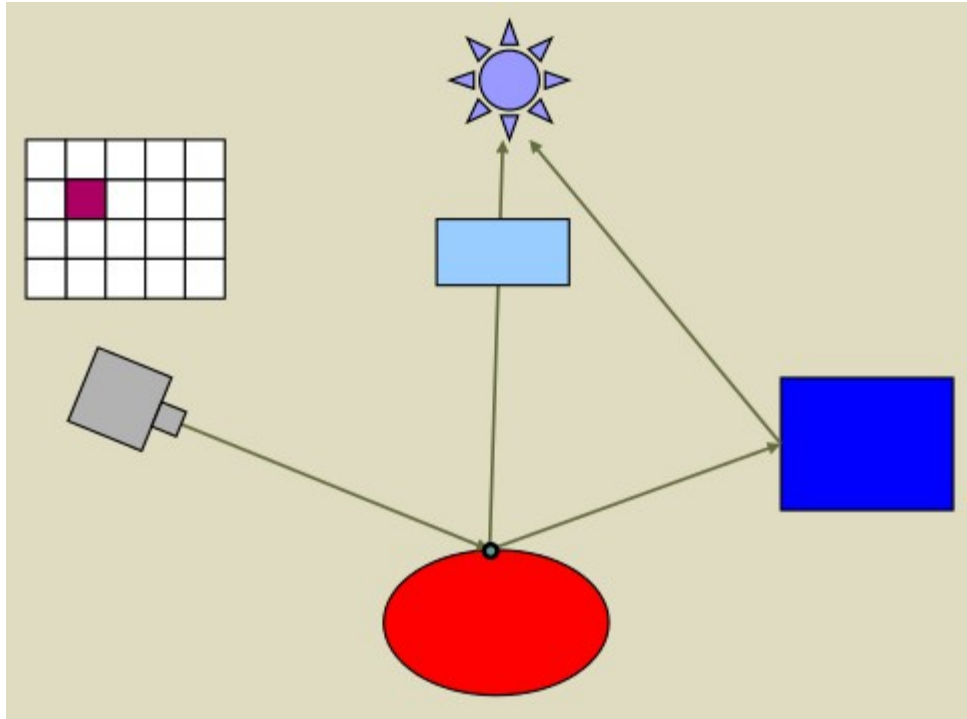
$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{Macierz translacji}$$

$$\mathbf{S} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{Macierz skalowania}$$

Obie te macierze są macierzami translacji. Pierwsza operacja dodaje nam do współrzędnej x 7, a druga dodaje do współrzędnej z 20. No i jeszcze te dwie operacje składamy. To raczej proste zadanie. hardkor będzie jak da coś z rotacją.

12. Co jest najbardziej wymagające obliczeniowo (czy tam najtrudniejsze, nie pamiętam) w ray tracingu? Jak to wpływa na wybór obiektów na scenie?

Źródła: 07_Wlledzenie_promieni.pdf, [http://en.wikipedia.org/wiki/Ray_tracing_\(graphics\)](http://en.wikipedia.org/wiki/Ray_tracing_(graphics))



1 minutowy wstęp do ray tracingu: w tej metodzie wypuszczamy promienie z kamery (trochę nieintuicyjne). Następnie te promienie się odbijają/załamują/pochłaniają i lecą dalej. Z tego wszystkiego obliczamy kolor danego piksela. Jest to fajna metoda, ale bardzo wolna - z tego powodu nie stosowana w grach.

Co jest najbardziej kosztowne obliczeniowo: "prowadzenie" tych promieni, ponieważ liczba promieni może nam bardzo szybko rosnąć. Potem dla każdego z tych promieni musimy **obliczać kolizje z obiektami**, wyznaczać kolory oraz tworzyć nowe promienie.

13. Jakie główne czynniki wpływają na czas renderingu metodą ray tracingu?

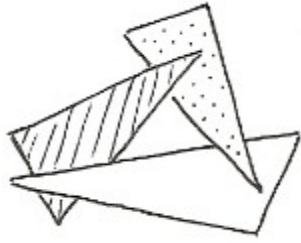
Obliczanie kolizji. Dlatego wymyślono te całe siatki i i obiekty otaczające

14. Co to jest tekstura parametryczna

15. Do czego służy bufor głębokości w OpenGL?

Źródło: http://pl.wikipedia.org/wiki/Bufor_Z, laboratoria, notatki (str. 8)

Inna nazwa "Bufor Z" - bufor (tablica dwuwymiarowa) która dla każdego piksela trzyma jego wartość współrzędnej **Z** (odległości od obserwatora) jako **integer**. Bufor głębokości nie przechowuje natomiast wartości RGB. No i teraz jak chcemy dodać jakiś obiekt do sceny to wiemy, gdzie będzie on widoczny a gdzie zasłonięty przez inne rzeczy, tak więc wiemy które piksele zupdatować.



problem: który trójkąt narysować
jako pierwszy?

rozwiązanie: dzielimy na mniejsze
trójkąty albo stosujemy
bufor głębokości

16. Jaki mechanizm pozwala na automatyczne zasłanianie obiektów i jaki parametr OpenGL do tego służy?

Chodzi o Bufor Z.

W pytaniu w tym roku było "przełącznik" - jak na moje to przełącznikiem jest `glEnable(GL_DEPTH_TEST)`; poniżej inne opcje, które też są wymagane, żeby to wszystko hulało:

```
glDepthMask(GL_TRUE);
glDepthFunc(GL_LEQUAL);
glDepthRange(0.0f, 1.0f);
```

Dodatkowo trzeba ustawić `initDisplayMode`:

```
void glutInitDisplayMode(unsigned long int mode);
```

`mode` jest bitową sumą poszczególnych masek bitowych (zapisanych symbolicznie), między innymi:

`GLUT_DEPTH` **aktywny bufor głębokości**

17. Co zyskujemy (w czym nam pomaga) używając funkcji `glNormalize()`

Źródło: <http://www.opengl.org/archives/resources/features/KilgardTechniques/oglpitfall/#1>

Internet mówi że jak przykładowo skalujemy sobie obiekt `glMatrixMode(GL_MODELVIEW); glScalef(3.0, 3.0, 3.0);`. To wtedy skalują nam też się normalne związane z wierzchołkami. No i wtedy się robi kicha, bo system oświetlania korzysta z długości normalnych. Tak więc zmienia nam się kolorki. Wywołanie `glEnable(GL_NORMALIZE)`; mówi OpenGLowi żeby sobie "znormalizował" normalne do długości 1 zanim będzie je brał pod uwagę w obliczaniu światła.

18.  się liczy normalną i czym się różni normalna w matematyce od OpenGL-owej?

Normalna w grafice komputerowej (w szczególności w OpenGL) nie musi być prostopadła do powierzchni, tak jak w matematyce. Wektory normalne mogą być zaczepione także w wierzchołkach.

źródło: Notatki, str. 7

Za tajemniczym źródłem: <http://samszyn.2ap.pl/1.pdf> :

OpenGL nie dostarcza nam żadnego mechanizmu do liczenia normalnych. My musimy to w programie sami zrobić, musimy wyznaczyć wektor prostopadły w jakimś miejscu do czegoś tam (do płaszczyzny do trójkąta) a zaczepiony w jakimś wierzchołku. Najlepiej policzyć iloczyn wektorowy.

Mamy dwa wektory, które są współpłaszczyznowe, są np. krawędziami jakiegoś trójkąta, mnożąc je wektorowo przez siebie uzyskujemy trzeci wektor, który jest prostopadły do obu tych wektorów czyli jest prostopadły do płaszczyzny, która te wektory wyznacza i ten wektor możemy potraktować jako wektor normalny do tej płaszczyzny.

Natomiast funkcja `GL_Normalize` nie liczy normalnej, natomiast jeżeli normalne są już policzone i są wskazane, że to są normalne to `GL_NORMALIZE` normalizuje je do jedynki. jeśli tego nie użyjemy to może się tak zdarzyć, że wyliczony przez nas wektor nie jest jednostkowy .

19. Jak w programie graficznym można wyświetlić teksturę 3D?

W programie takim nakłada się teksturę na model 3D i dopiero potem można po niej rysować.

Przykład z Photoshopa:

http://help.adobe.com/pl_PL/photshop/cs/using/WS0BA787A7-E4AC-4183-8AB7-55440C51F95B.html#WS58DF666E-25D7-43e5-9CB1-F54A989EAF70



Niestety więcej pytań nie znalazłem, wykopalem całe forum i wiki. Widocznie to jakaś nowa rozrywka Aldy.

ZAGADNIENIA OD ALDY:

1. ELEMENTY OPENGL

- Jaką techniką w OpenGL jest realizowane automatyczne zasłanianie jednych obiektów przez inne? Jaki przełącznik je umożliwia?

<https://docs.google.com/document/d/1M0fQv1I1T5lXwd1iSMDJxcLMCap6AIHCFfPd-cl3vyA/edit#bookmark=id.ndcxu7538htz>

- Czego dotyczy w OpenGL podwójne buforowanie obrazu?

Typowo OpenGL stosuje dwa buforory koloru: przedni i tylny. Bieżąca scena znajduje się w przednim buforze i jest wyświetlana na ekranie monitora, a jednocześnie w tylnym buforze rysowana jest kolejna ramka sceny. Po zakończeniu rysowania, buforory są zamieniane i cały proces zaczyna się od początku. Podwójne buforowanie eliminuje migotanie obrazu występujące przy stosowaniu pojedynczego bufora koloru.

- Do czego służy bufor VBO?

Za Wikipedią: Vertex Buffer Object (VBO) - rozszerzenie [OpenGL](#) umożliwiające tworzenie obiektów zawierających opis geometrii. Rozszerzenie to umożliwia szybkie renderowanie sceny i znacznie zwiększa wydajność aplikacji.

W uproszczeniu, polega na jednorazowym przekazaniu sterownikowi graficznemu [tablic danych](#), zawierających współrzędne wierzchołków, wektory normalne, mapowanie tekstury, kolory itp., a następnie posługiwaniu się numerami tych tablic podczas renderowania poszczególnych obiektów w kolejnych klatkach.

A za wykładem: Tradycyjne podejście w wersji ≤ 2.1 zakładało definiowanie każdego wierzchołka oddzielnie za pomocą funkcji:

```
glVertex(x,y,z)

umieszczonej między wywołaniami:

glBegin(mode);

    glVertex(...);

    glVertex(...);



...

glEnd();
```

Taki styl jest **przejrzysty**, ale **nieefektywny**. W rzeczywistości każdy wierzchołek jest przesyłany oddzielnie, a to kosztuje. Obecnie staramy się grupować wszystkie wierzchołki i przysyłać je łącznie do karty graficznej. **Wierzchołki są przekazywane do bufora jako obiekt (VBO)** i przetwarzane przez *vertex shader*.

- **Do czego służą zmienne typu Uniform?**

uniform - opisuje wartości, które nie zmieniają się w trakcie renderowania (czyli takie które charakteryzują otoczenie, a nie renderowane obiekty, np. może to być położenie źródła światła lub kolor światła). W związku z tym mają atrybut *read-only*. Zmienne *uniform* są dostępne w *vertex* i *fragment shaderach*.

-  **wektory w GLSL**
 - wektorowe - *vec2*, *vec3*, *vec4*
 - macierzowe - *mat2*, *mat3*, *mat4*
ogólnie:  $\{wiersze \times kolumny\}$
 $2 \leq wiersze, kolumny \leq 4$

W nowszych wersjach GLSL doszedł typ podwójnej precyzji (wektorowy i macierzowy też).

- Rodzaje shaderów.

Według kolejności wykonywania:

Vertex shader operuje na wierzchołkach. Może wykonywać na nich:

- transformacje położenia wierzchołków (z pomocą macierzy widoku i rzutowania)
- transformacje normalnych (łącznie z ich normalizacją)
- transformacje współrzędnych tekstur
- obliczanie oświetlenia w wierzchołkach
- obliczanie koloru w wierzchołkach

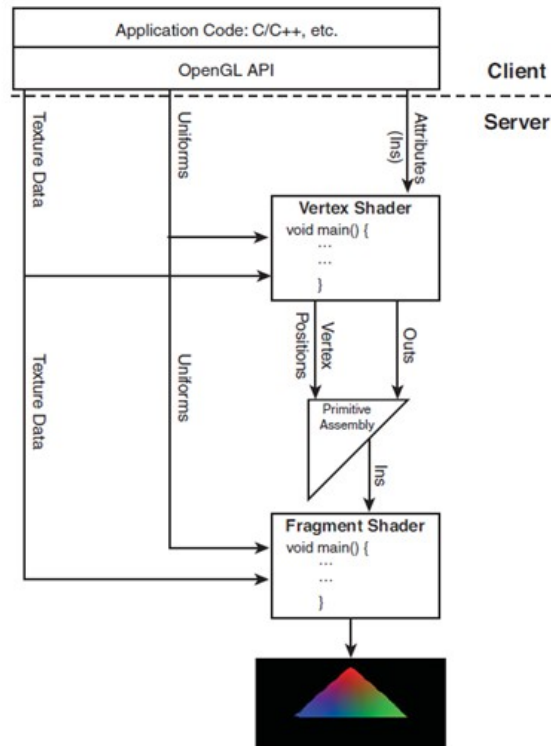
Wierzchołki przetwarzane są pojedynczo i niezależnie.

Geometry shader (opcjonalny) przetwarza grupy wierzchołków, które tworzą podstawowe elementy geometryczne (trójkąty, łamane, odcinki, punkty). Ma dostęp do całej grupy wierzchołków, może zmieniać typ i atrybuty elementu geometrycznego, może dodawać i kasować wierzchołki.

Fragment shader działa na fragmentach (pikselach), czyli na obrazie powstałym na skutek rasteryzacji. Na fragmentach może wykonywać:

- obliczanie kolorów i współrzędnych tekstur per pixel
- obliczanie mgły
- obliczanie normalnych dla oświetlenia per pixel
- **nie może** zmienić współrzędnych piksela

- Schemat potoku graficznego.



Potok renderowania w uproszczeniu; model klient-serwer. Klient - kod pracujący na CPU, serwer - kod pracujący na GPU. Na uproszczonym rysunku zaznaczone są dwa shadery, które stanowią minimum: *Vertex Shader* i *Fragment Shader*. Trzeci, *Geometry Shader*, jest tu pominięty.

2. TRANSFORMACJE

- Jak reprezentowane są transformacje?

Transformacje są reprezentowane jako macierze 4x4. Dokonać transformacji jakiegoś wierzchołka możemy poprzez pomnożenie jego współrzędnych (wektor czteroelementowy) przez taką macierz transformacji. W wyniku mnożenia dostajemy przetransformowany wierzchołek. Przykład:

$$\begin{bmatrix} x' \\ y' \\ z' \\ w' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 10 \\ 0 & 1 & 0 & 10 \\ 0 & 0 & 1 & 10 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} x+10w \\ y+10w \\ z+10w \\ w \end{bmatrix}$$

- Dlaczego macierze 4x4?

Dlaczego mamy współrzędne x , y , z jest raczej oczywiste. Dodatkowa zmienna w jest tutaj swoistym akumulatorem. Jest ona nam potrzebna, ponieważ po wykonaniu ciągu

transformacji nowy wierzchołek nie ma od razu odpowiednich współrzędnych. Otrzymujemy je dopiero po podzieleniu odpowiednio x, y i z przez wynikowe w (znormalizowanie współrzędnych).

http://en.wikipedia.org/wiki/Transformation_matrix#Perspective_projection

Wg wykładów Sokoła: "rozmiary macierzy są o 1 większe by móc ująć w niej wyrazy wolne zachowując schemat przekształceń $v' = Mv$." cokolwiek to znaczy...

Gdy dodamy wymiar do macierzy, to zamiast wykonywać operacje mnożenia i dodawania, możemy wykonać jedynie operacje mnożenia, co pozwala nam na składanie przekształceń. Np zamiast:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = A \cdot X + T = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y + t_x \\ a_{21}x + a_{22}y + t_y \end{bmatrix}$$

Możemy zapisać:

$$\begin{bmatrix} x' \\ y' \\ W' \end{bmatrix} = A \cdot X = \begin{bmatrix} a_{11} & a_{12} & t_x \\ a_{21} & a_{22} & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ W \end{bmatrix} = \begin{bmatrix} a_{11}x + a_{12}y + t_xW \\ a_{21}x + a_{22}y + t_yW \\ W \end{bmatrix}$$

Wierzchołki w grafice komputerowej zapisywane są we współrzędnych jednorodnych, tj punkty w przestrzeni n wymiarowej zapisujemy jako n+1 wymiarowe. Nadają się one idealnie do opisu przekształceń w przestrzeniach n-wymiarowych, bo łatwiej jest wtedy wykonywać przekształcenia. W 3-wymiarowej przestrzeni mamy 4 - wymiarowy punkt, dlatego też mamy macierze transformacji o wielkości 4x4.

Przykład:

http://pl.wikipedia.org/wiki/Wsp%C3%B3%C5%82rz%C4%99dne_jednorodne

- Podstawowe macierze transformacji

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{Macierz translacji}$$

$$\mathbf{S} = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \text{Macierz skalowania}$$

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) & 0 \\ 0 & \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$R_y = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

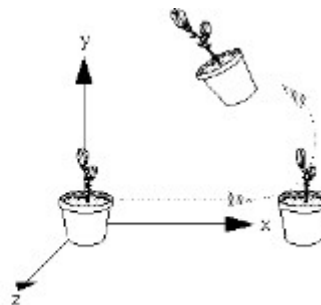
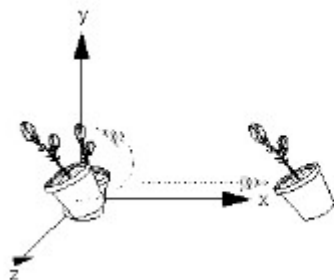
$$R_z = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Macierze
obrotów

- Kolejność składania transformacji.

Kolejność wykonywania transformacji ma oczywiście znaczenie. Składanie transformacji następuje zgodnie z kolejnością wykonywania mnożenia (od lewej do prawej), czyli przykładowo stosując powyższe oznaczenia macierzy: $v \cdot R \cdot T \cdot S$ oznacza, że najpierw nastąpi obrót, potem translacja i na końcu skalowanie.

Oto wiele mówiący przykład na obrazku:



Dla nieruchomego układu współrzędnych:

```
LoadIdentity();
MultMatrixf(T); /* translation */
```

```

MultMatrixf(R); /* rotation */
draw_the_object();

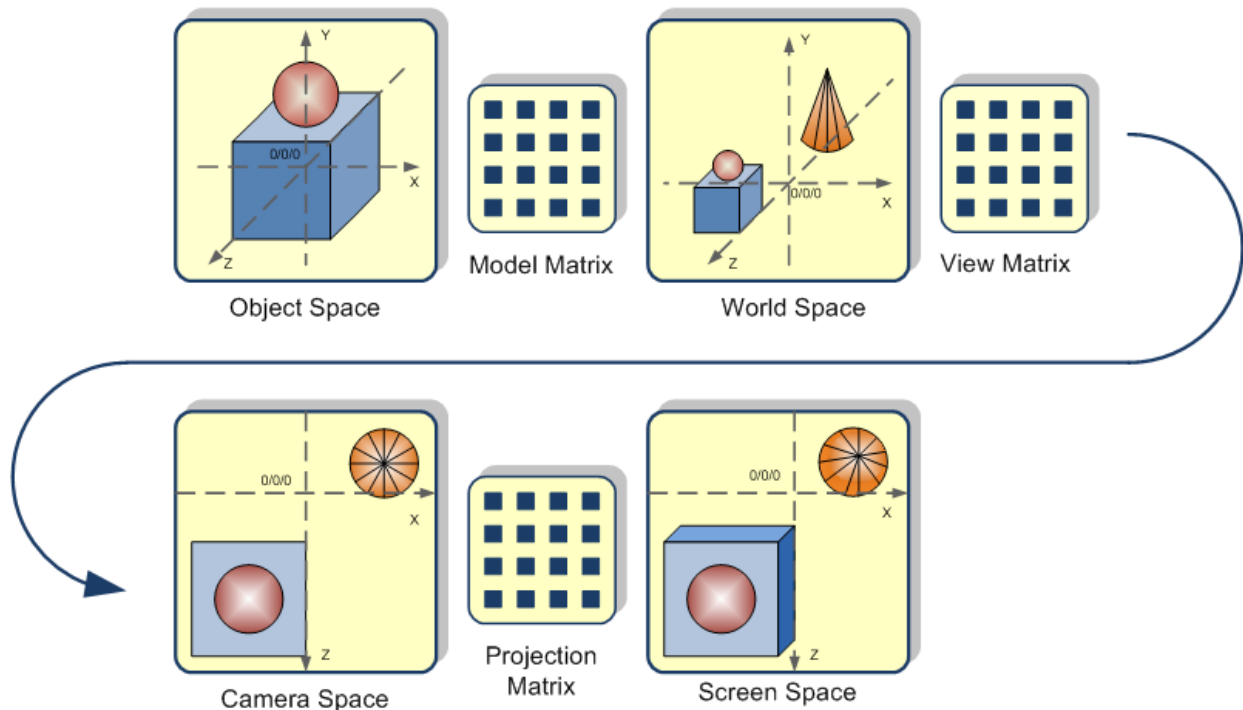
```

W logicznej analizie transformacji na wierzchołki najpierw działamy rotacją R , a następnie translacją T . Kolejność mnożenia macierzy jest jednak odwrotna!

A w układzie lokalnym:

Zapis instrukcji pozostaje taki sam, ale interpretacja jest inna. W lokalnym układzie najpierw wykonujemy translację, potem rotację.

- Typowe ustawienia układów współrzędnych: model space, camera space, world space, clip space



Trochę inne nazewnictwo tutaj jest używane. Zamiast Model space jest Object space, zamiast clip space jest screen space.

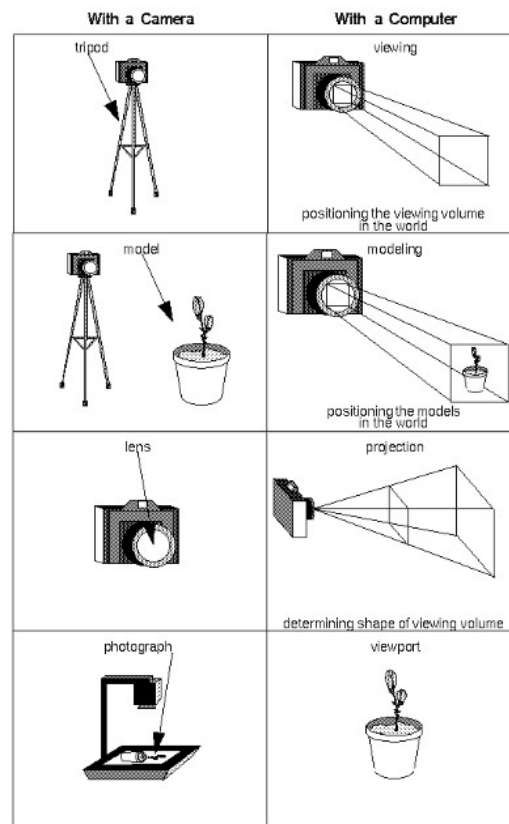
Po kolei:

- **model space** (object space) to układ współrzędnych dotyczący tylko jednego obiektu lub grupy obiektów (np. modelujemy sześcian składając go z 8 wierzchołków i nadajemy im współrzędne $[\pm 1, \pm 1, \pm 1]$. Nie obchodzą nas w tej chwili inne obiekty.
- **world space** (niektórzy mówią model space...) to układ, w którym bierzemy już pod uwagę wzajemne położenie obiektów. model matrix zawiera właśnie te informacje w postaci odpowiednich translacji, które trzeba wykonać

- **camera space** (eye space, view space) - teraz bierzemy pod uwagę to, że na powyższy świat patrzymy z jakiejś perspektywy. Zgodnie z jakimś tam prawem poruszanie kamerą jest równoważne do poruszania wszystkimi obiektami i to wykorzystujemy tworząc view matrix. Teraz wszystkie współrzędne odnoszą się do kamery (oka)
- **clip space** (screen space) - ostatni układ do ten docelowy, dwuwymiarowy, którym posługuje się już monitor. Nie jesteśmy w stanie jeszcze wyświetlać 3D i trzeba wszystko rzutować na płaszczyznę. Za to odpowiada projection matrix. Są różne rodzaje takiego rzutowania oddające różne właściwości, w zależności na czym bardziej nam zależy (nigdy nie uzyskamy tego samego efektu co w 3d)

źródło: <http://www.matrix44.net/cms/notes/opengl-3d-graphics/coordinate-systems-in-opengl>

Mam wrażenie, że te zagadnienia były omówione na wykładzie pod jeszcze innymi nazwami:



Viewing - Transformacje widoku określają położenie kamery


Modeling - Transformacje modelowania przemieszczają obiekty na scenie (Modelview - określa wzajemne relacje Viewing i Modeling)

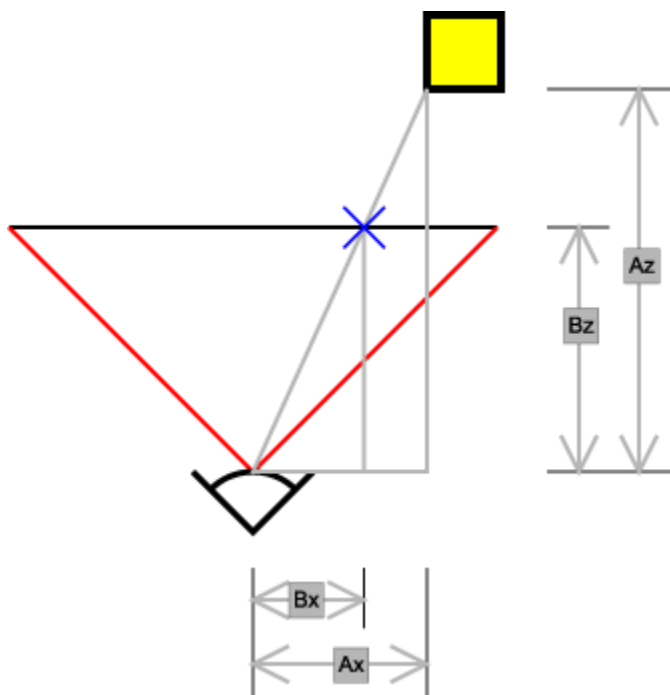
Projection - Transformacje rzutowania definiują bryłę widoku i płaszczyznę obcięcia

Viewport - Mapowanie na okno

Generalnie na jaką stronę by się nie weszło to zawsze są inne nazwy...

- **rodzaje rzutowania**

- **rzut prostokątny** (orthographic projection) - jeden z rodzajów rzutu  prostopadłego, linie projekcji podczas rzutowania są równoległe (stąd nazwa angielska), przecinają prostopadłe płaszczyzne na którą rzutujemy (stąd polska nazwa). Rzutowanie takie nie wnosi żadnej poprawki na perspektywę i nie jest naturalne dla człowieka. Używa się go między innymi w CADach, bo nie zmienia rozmiaru obiektu bez względu na odległość od kamery. Czasem używa się tego też w grach 2D oraz do gier z rzutem izometrycznym (w opengl chyba nie ma takiego rzutu out of the box)
- **rzut perspektywiczny** (perspective projection) - rzut który zachowuje wrażenie perspektywy, czyli pozornego zmniejszania się obiektów w zależności od odległości. Uwzględnia też takie rzeczy jak pole widzenia. Jest naturalny dla ludzkiego oka. Poniższy rysunek przedstawia idee w uproszczeniu (tutaj rzutujemy 2D na 1D):



Więcej informacji: http://en.wikipedia.org/wiki/Perspective_projection

3. OŚWIETLENIE

- Dlaczego w OpenGL w modelu ADS wprowadza się kilka rodzajów oświetlenia (ambient, diffuse, specular), podczas gdy w rzeczywistości mamy tylko jeden rodzaj światła?

Dokładne odwzorowanie rzeczywistego oświetlenia w komputerze jest praktycznie niemożliwe, stosujemy więc uproszczenia.
Żeby powstało nam coś wygląda sensownie, to kombinujemy z tymi uproszczonymi.

Tu po kolei dodajemy do ambient najpierw diffuse a potem specular:



i wychodzi fajna kulka.

- Znaczenie składowych ambient, diffuse i specular.

<https://docs.google.com/document/d/1M0fQv1I1T5IXwd1iSMDJxcLMCap6AIHCFfPd-cl3vyA/edit#bookmark=id.acz1s2gcn8jp>

- Wektory normalne - do czego służą i jak je liczymy?

Mamy dane 2 wektory, które znajdują się na wspólnej płaszczyźnie (np. są krawędziami jakiegoś wielokąta - tudzież: trójkąta). Obliczając ich iloczyn wektorowy, otrzymujemy wektor, który jest prostopadły do obu tych wektorów, czyli jest prostopadły do płaszczyzny, którą wyznaczają. Możemy go potraktować jako wektor normalny do tej płaszczyzny.

Wektor normalny określa również **stronę** powierzchni.

Ogólnie trójkąty mają stronę wierzchnią i spodnią. Wektor normalny pokazuje nam, gdzie jest “wierzch” trójkąta. Np. jak mamy czajnik, to oblicza się tylko to jak on wygląda z zewnątrz, a to co w środku czajnika się dzieje nikogo nie obchodzi.

//raczej chodzi o to: skąd po wyliczeniu iloczynu wektorowego mamy pewność, że aktualnie wyliczona strona ‘zewnątrz’ jest tą, którą chcemy żeby była na zewnątrz?

Wektory normalne mogą być generowane przez program, który nam wyprodukował obiekt (razem z wierzchołkami i współrzędnymi tekstur), albo możemy je sami wyliczać np. w sposób podany powyżej dla każdego trójkąta. Potrzebne są do wyliczania oświetlenia diffuse - robimy iloczyn skalarny znormalizowanych wektorów: normalnego i przeciwnego do kierunku padania światła - wychodzi nam jasność. Jak wiadomo, w iloczynie skalarnym mamy cosinus kąta pomiędzy wektorami - im bardziej ‘prostopadle’ źródło światła świeci na dany trójkąt, tym

jasniejszy on będzie. Potrzebne są też w oświetleniu specular do wyliczenia promienia odbitego.

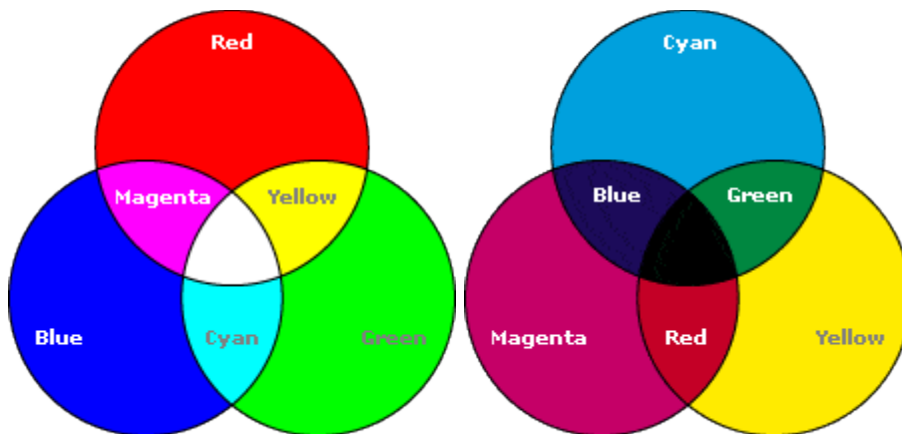
- **Lokalne, a globalne modele oświetlenia.**

Lokalny model jest wtedy, gdy poszczególne obiekty nie wpływają na siebie podczas procesu cieniowania. Takie modele są o wiele szybsze ale pozbawiają nas pełnego realizmu.

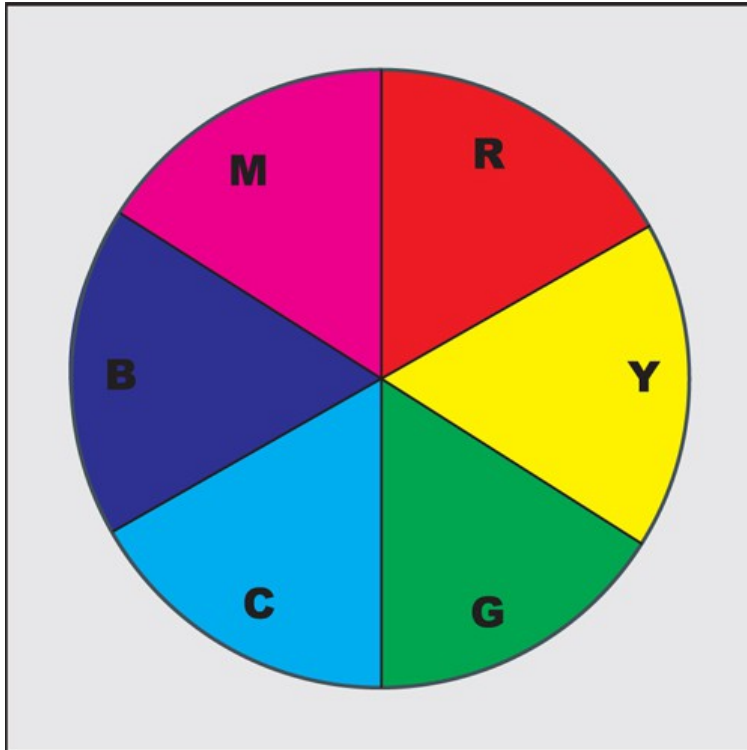
Modele globalne natomiast wykorzystują pełną informację o scenie, dzięki czemu możliwe jest wygenerowanie oświetlenia nawet dla miejsc, które w modelach lokalnych byłyby oświetlone słabo / wcale (na przykład piłka po stronie obserwatora, gdzie światło jest za piłką a znajdujemy się w pokoju) .

Ray tracing, path tracing, photon mapping i Radiosity to modele globalne, a np. model Phong'a jest lokalny(?) //tak, jest lokalny, tak samo jak Blinn-Phong'a

- **Barwa światła, parametry materiałowe, mieszanie barw.**



Na tej palecie barwnej macie połączone 2 powyższe:
http://www.optyczne.pl/upload2/78991_08-4-09.jpg



Własności materiałowe obiektów – określają jak odbija się światło, w tym kolor światła odbitego

- **Oświetlenie liczone per vertex i per pixel - czym się różnią?**

Za wykładem:

Przeciwnym podejściem w stosunku do ustalania stałej barwy/jasności dla całego wielokąta jest wyliczanie oświetlenia dla każdego punktu oddzielnie. Jest to jednak kosztowne obliczeniowo i niejasne – co to znaczy „każdy punkt”

Wszystkie pozostałe męczone na wykładzie -> per vertex + interpolacja (bo ostatecznie musimy mieć wszędzie) (choć w sumie nie wiem czy raytracing jest per vertex czy per pixel...) raczej per piksel, bo liczysz promienie wychodzące z ekranu, a gdzie one trafią to już inna sprawa

- **Interpolowanie oświetlenia.**

shading (cieniowanie) jest określeniem metody determinującej jasność i barwę punktu na powierzchni.

rodzaje cieniowania lokalnego:

- flat shading - stałą wartością
Dobrze i szybko działa, gdy albo źródło światła jest bardzo daleko albo kamera jest bardzo daleko od obiektu
- Gouraud shading - interpolacja jasności i barwy
<https://docs.google.com/document/d/1M0fQv1I1T5IXwd1iSMDJxcLMCap6AIHCFfPd-cl3vyA/edit#bookmark=id.7v4y3i2ciqsu>

- Phong interpolation - interpolacja wektora normalnego (patrz niżej)

◦ Model Phong'a oświetlenia połyskliwego

Porównanie z Gouraudem:

<https://docs.google.com/document/d/1M0fQv1I1T5IXwd1iSMDJxcLMCap6AIHCFfPd-cl3vyA/edit#bookmark=id.kxd2spn717x9>

◦ Idea metody śledzenia promieni

<https://docs.google.com/document/d/1M0fQv1I1T5IXwd1iSMDJxcLMCap6AIHCFfPd-cl3vyA/edit#bookmark=id.hqjjm3oa9lc>

◦ Idea metody energetycznej

Radiosity: metoda teoretycznie lepsza od Ray Tracingu - daje bardzo realistyczne obrazy, uwzględnia tylko światło rozproszone. Obliczenia są niezależne od położenia obserwatora, końcowy wynik to model 3-D. W metodzie radiosity nie uzyskamy odbłasków!!!

Podstawowe założenia:

- powierzchnie są przedstawiane jako oddzielne wielokąty (patches)
- wielokąt jest oświetlony z jednakową intensywnością na całej powierzchni
- Model opisuje przekaz światła (energii) pomiędzy elementami przy pomocy układu równań liniowych
- Rozwiązanie układu określa natężenie oświetlenia każdego elementu

Chcemy obliczyć jak mocno jest oświetlony dany wielokąt. Używamy do tego magicznego wzorku

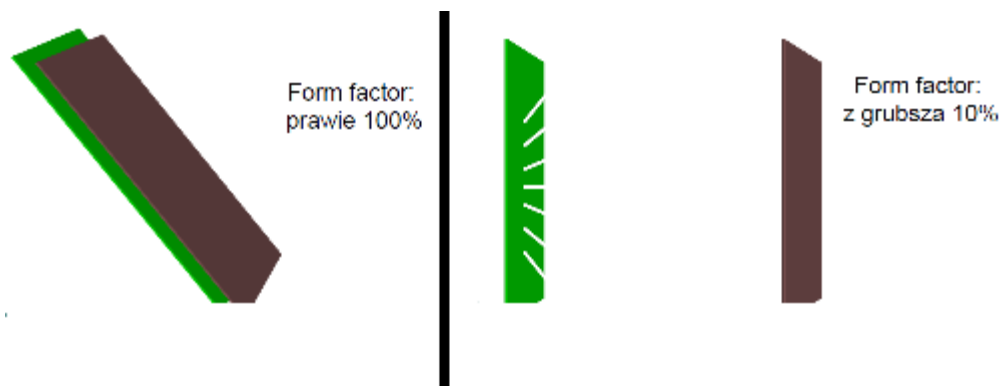
$$B_i = E_i + \rho_i \sum_j B_j F_{ji} (A_j / A_i)$$

który mówi nam, że natężenie oświetlenia (B) dla wielokąta i (dla którego sobie to wszystko liczymy) można uzyskać sumując dwie rzeczy

1) światło, którym świeci sam (E to energia/powierzchnia/czas)

2) światło, które dociera do niego od wszystkich innych pomnożone w odpowiednich miejscach przez współczynniki odbicia elementu i (ρ_i) o współczynnik sprzężenia (F)

Współczynnik sprzężenia mówi nam o tym, jak bardzo elementy i i j wpływają na siebie. Np:



“A” to pole, i przy odbiciu rozproszonym ładnie się skraca i zostajemy ze wzorkiem:

$$B_i - \rho_i \sum B_j F_{ij} = E_i$$

Wikipedia może się przydać do obejrzenia jakie to efekty daje.

[http://en.wikipedia.org/wiki/Radiosity_\(computer_graphics\)](http://en.wikipedia.org/wiki/Radiosity_(computer_graphics))

Pytania z tegorocznego kolosa w terminie “0”:

1. [1. Jaką techniką w OpenGL jest realizowane automatyczne zasłanianie jednych obiektów przez inne? Jaki przełącznik je umożliwia?](#)
2. [2. Dlaczego w OpenGL w modelu ADS wprowadza się kilka rodzajów oświetlenia, gdy w rzeczywistości mamy jeden rodzaj światła?](#)
3. [3. Co jest najbardziej wymagające obliczeniowo w Ray Tracingu? Jak to wpływa na wybór obiektów na scenie?](#)
2. 4. [4. Jak się liczy normalną i czym się różni normalna w matematyce od tej w OpenGL?](#)
5. [5. Było podane działanie na macierzach i trzeba było napisać jaką transformację to przedstawia.](#)
6. [6. Globalny model oświetlenia.](#)
7. [7. Jaki kolor kuli uzyskamy, jeżeli będziemy oświetlać żółtą, czerwoną kulę zielonym światłem \(diffuse?\)?](#)
8. [8. Czego dotyczy interpolacja Gourauda?](#)
9. [9. Wzór w interpolacji Phong \(jakoś tak\).](#)
10. [10. Do czego służą zmienne typu Uniform?](#)