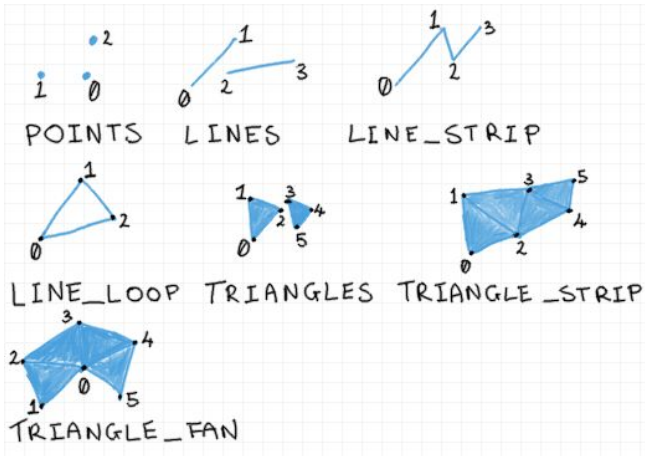
1 ELEMENTY WEBGL

1.1 Co to są wierzchołki?

- są podstawą rysowania
- mogą służyć do reprezentowania obiektów
- znacznie wygodniej jest na podstawie wierzchołków zbudować siatkę trójkątów
- w OpenGL wierzchołek jest zbiorem atrybutów
    - Podstawowe atrybuty:
        - współrzędne położenia
        - barwa
        - kierunek wektora normalnego
        - współrzędne tekstury nakładanej
- wierzchołki organizowane są w tablice
- obecnie staramy się grupować wszystkie wierzchołki i przesyłać je łącznie do karty graficznej

1.2 Jakie podstawowe elementy graficzne możemy rysować w WebGL?

- POINTS
- LINES
- LINE_LOOP
- LINE_STRIP
- TRIANGLES
- TRIANGLE_STRIP
- TRIANGLE_FAN



1.3 Jaką techniką w OpenGL/WebGL jest realizowane automatyczne zasłanianie jednych obiektów przez inne?

Poprzez DEPTH_TEST. Każdy piksel trzyma wartość Z (Bufor Z, Bufor głębokości) (odległość od obserwatora). Podczas rysowania sceny uwzględniane są tylko te piksele, które nie będą zasłonięte przez inne.

1.4 Do czego służą zmienne typu 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.

1.5 Nowe typy zmiennych w GLSL

- wektorowe - vec2, vec3, vec4
- macierzowe - mat2, mat3, mat4
- plus typy podwójnej precyzji

1.6 Rodzaje shaderów.

Według kolejności wykonywania:
Vertex shader operuje na wierzchołkach. Może wykonywać na nich:

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

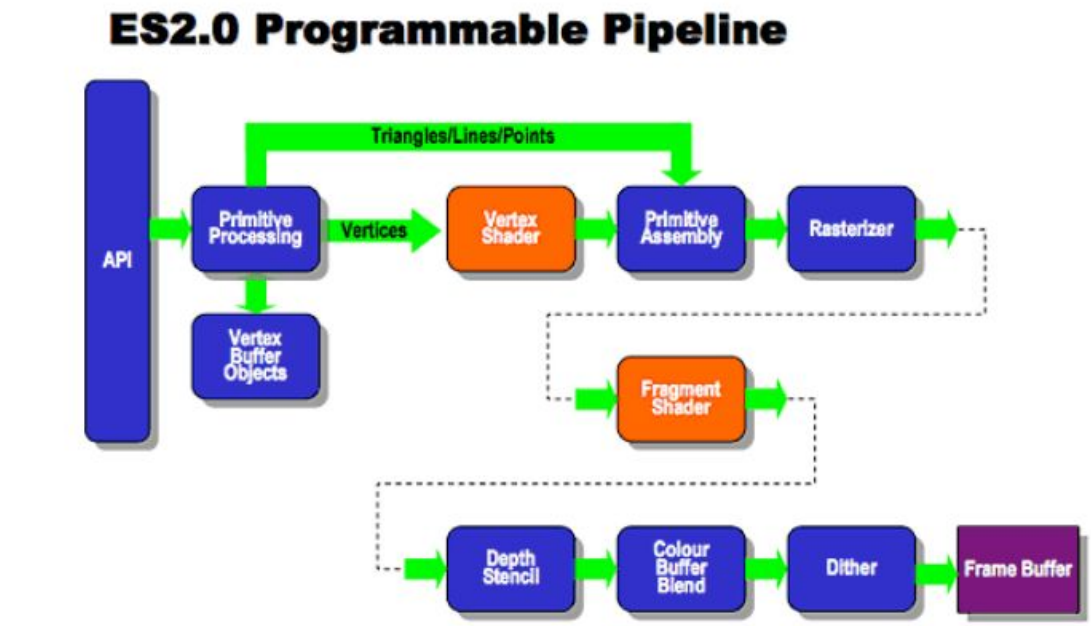
Wierzchołki przetwarzane są pojedynczo i niezależnie.
Geometry shader (opcjonalny) przetwarza grupy wierzchołków, które tworzą podstawowe elementy geometryczne (trójkąty, łamane, odcinki, punkty). Ma dostęp do całej grupy wierzchołków,

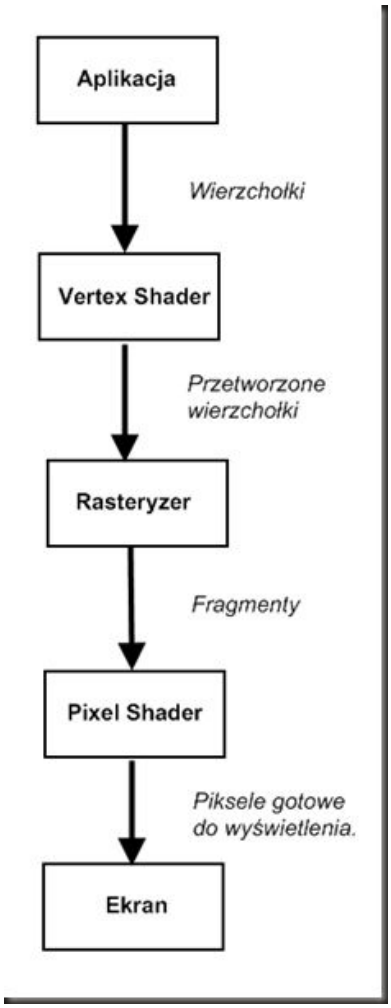
- może zmieniać typ i atrybuty elementu geometrycznego,
- może dodawać i kasować wierzchołki.

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

1.7 Schemat potoku graficznego.





2 Transformacje

2.1 Jak reprezentowane są transformacje?

Jako macierze 4x4? :)

2.2 Dlaczego macierze 4x4?

“rozmiary macierzy są o 1 większe by móc ująć w niej wyrazy wolne zachowując schemat przekształceń v’ = Mv.”

2.3 Podstawowe macierze transformacji

- Translacji
- Skalowania
- Obrotów

2.4 Kolejność składania transformacji.

Kolejność wykonywania transformacji ma oczywiście znaczenie. Składanie transformacji następuje odwrotnie niż kolejność wykonywania mnożenia (od lewej do prawej), czyli przykładowo stosując powyższe oznaczenia macierzy: v*R*T*S oznacza, że najpierw nastąpi skalowanie, potem translacja i na końcu obrót.

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

http://www.matrix44.net/cms/notes/opengl-3d-graphics/coordinate-systems-in-opengl
http://learnopengl.com/#!Getting-started/Coordinate-Systems

- **Object** - współrzędne zależą tylko od jednego obiektu, określone względem jego ‘originu’, czyli takiego środka ciężkości / uchwytu
- **Model = world (??)** - względem całej sceny, wszystkie obiekty
- **Camera** - względem obserwatora/kamery (*That way all the objects in the world are forward 10 units, making it look like you are backwards 10 units.*) ruch kamery -> ruch obiektów
- **Clip** - obcinanie współrzędnych spoza ustalonego zakresu (??)

--

- **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 wspórzędne [+/- 1, +/- 1, +/- 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, ktore 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 to 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)

2.6 Rodzaje rzutowania

- **Rzut prostokątny** (orthographic projection) - jeden z rodzajów rzutu prostopadłego, linie projekcji podczas rzutowania są równoległe (stąd nazwa angielska), przecinają prostopadle płaszczyzne na którą rzutujemy (stąd polska nazwa). Rzutowanie takie nie wnosi żadnej poprawki na perspektywe i nie jest naturalne dla człowieka. Używa się go miedzy 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.

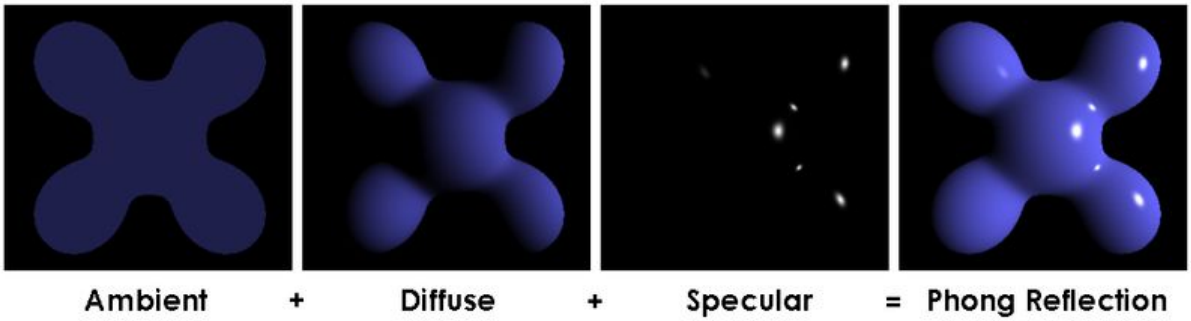
3 Oświetlenie i tekstury

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

Wynika to z tego, że radykalnie upraszczamy model oświetlenia (szybkość obliczeń). W OpenGL nie uwzględniamy odbić od obiektu do obiektu, mamy tylko odbicia źródła światła od obiektu do obserwatora.

3.2 Znaczenie składowych ambient, diffuse i specular.x

- **Ambient**: światło otaczające - nie pochodzi z żadnego określonego kierunku. Ma swoje źródło, jednak promienie światła odbijają się po całym pomieszczeniu lub scenie i generalnie są pozbawione kierunku. Obiekty oświetlone w ten sposób są równomiernie oświetlone na wszystkich powierzchniach we wszystkich kierunkach
- **Diffuse:** światło rozproszone - pochodzi z konkretnego kierunku, lecz jest odbijane od powierzchni równomiernie. Nawet jeśli światło jest odbijane równomiernie, powierzchnia jest jaśniejsza, gdy światło pada na nią bezpośrednio, niż wtedy, gdy pada na nią pod większym kątem
- **Specular:** światło odbłysków - podobnie jak światło rozproszone, posiada kierunek, ale jest odbijane ostro i w jedną stronę. Bardziej pobłyskujące obiekty możemy poznać po jasnych, lśniących plamach światła na ich powierzchniach



3.3 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.

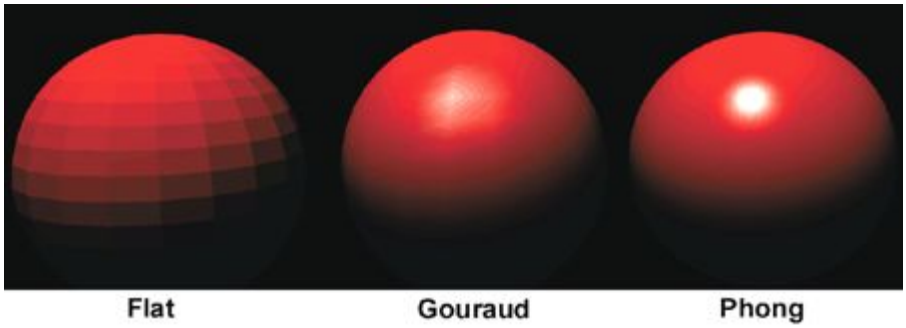
Wektory normalne mogą być generowane przez program, który nam wyprodukował obiekt (razem z wierzchołkami i współrzędnymi tekstur), albo możemy je wyliczać w sposób podany powyżej dla każdego trójkąta. Potrzebne są do wyliczania oświetlenia diffuse - robimy iloczyn skalarny znormalizowanych wektorów: 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.

Jeśli w wierzchołkach policzymy wektor normalny jako średnią wektorów sasiadujących płaszczyzn, to możemy uzyskać wygładzenie powierzchni obiektu bez zagęszczania siatki trójkątów.

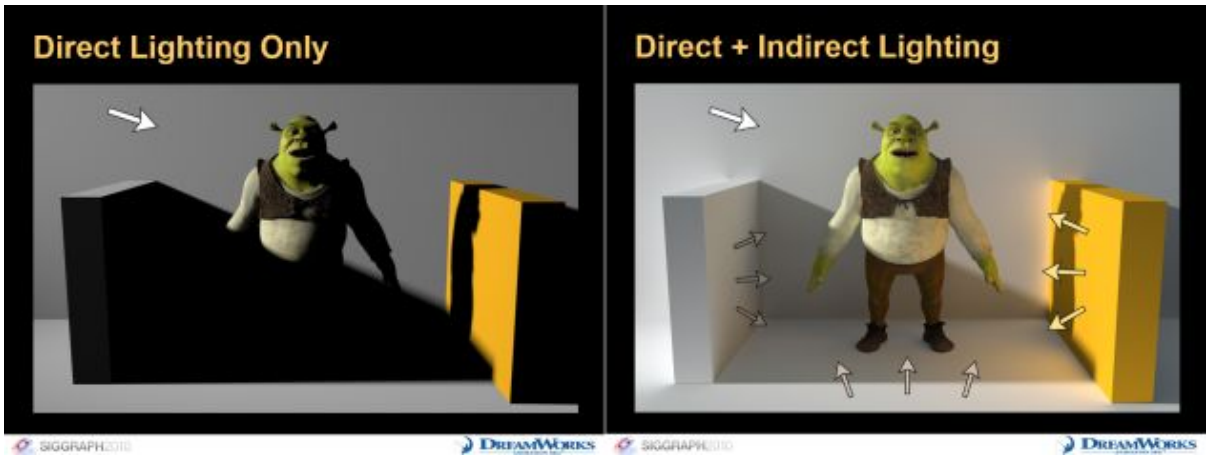
3.4 Lokalne, a globalne modele oświetlenia.

- **Oświetlenie lokalne:** obiekty traktowane są jako niezależne, nie uwzględniamy odbić światła, są szybkie i proste w obliczeniach, nie dają realizmu, nie wymagają wiedzy o całej scenie
    - **Modele:**
        - flat shading - Cieniowanie stałą wartością- Każdy wielokąt oświetlony jest ze stałą intensywnością i barwą na całej swojej powierzchni.
        - Gouraud shading - Cieniowanie z interpolacją jasności i barwy - Interpolacja jasności/barwy odbywa się na podstawie wartości wyznaczonych w wierzchołkach wielokątów, musimy więc znać normalne w wierzchołkach

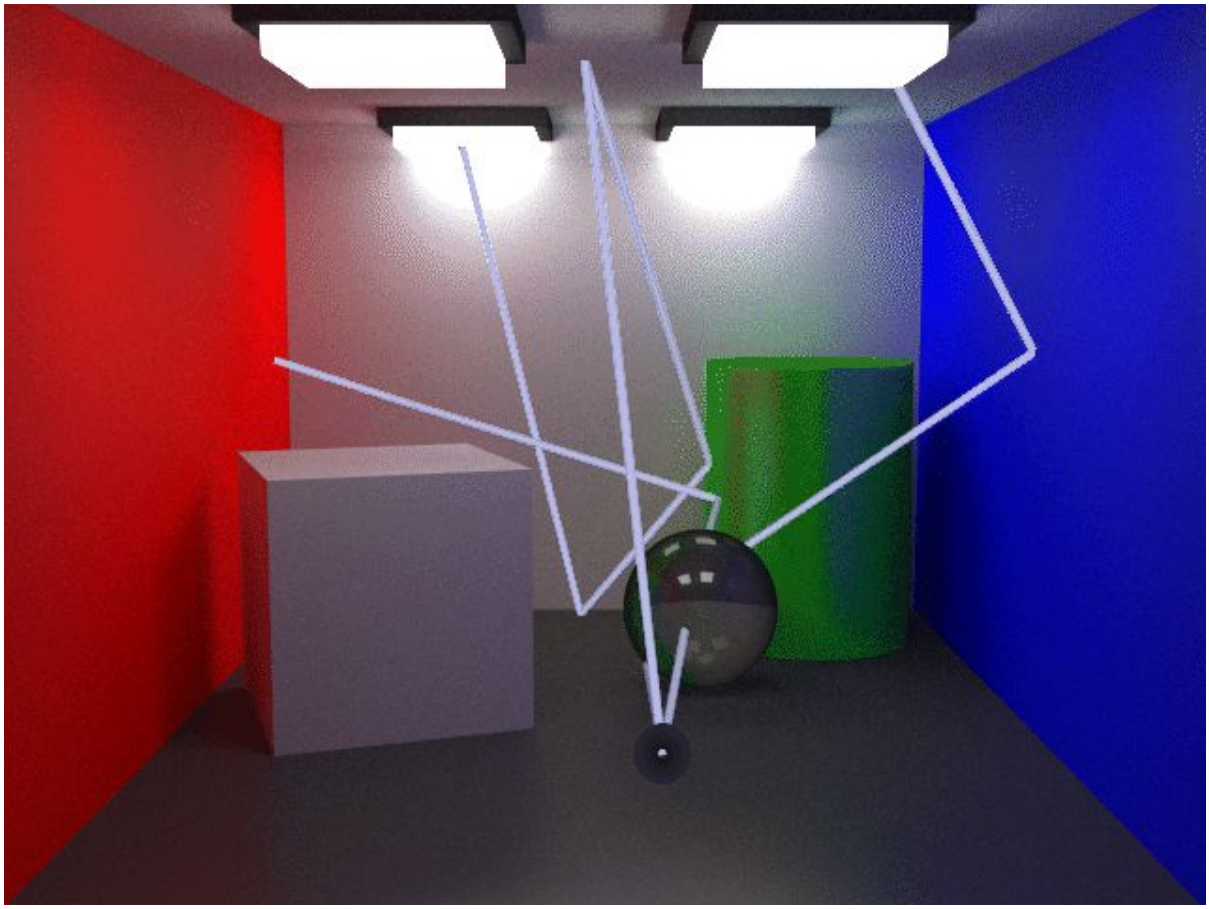
- Phong interpolation - Cieniowanie z interpolacją wektora normalnego - Opiera się na interpolacji normalnych sąsiadujących wielokątów.



- **Oświetlenie globalne:** śledzenie promieni? Wikipedia
  - **Modele:**
    - Mapowanie fotonowe
    - Path tracing
    - Radiosity

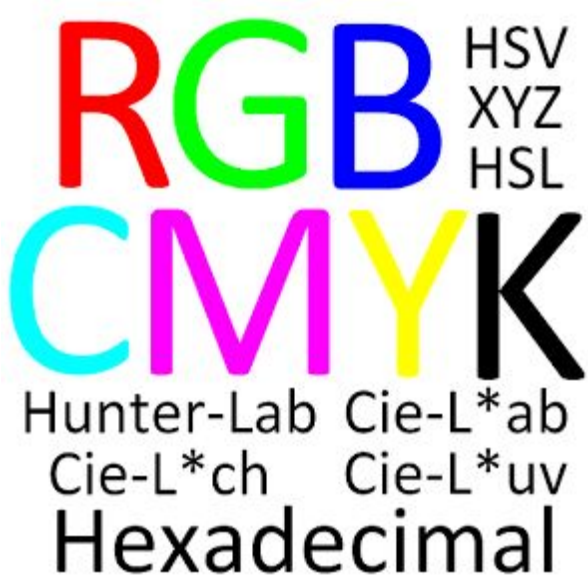


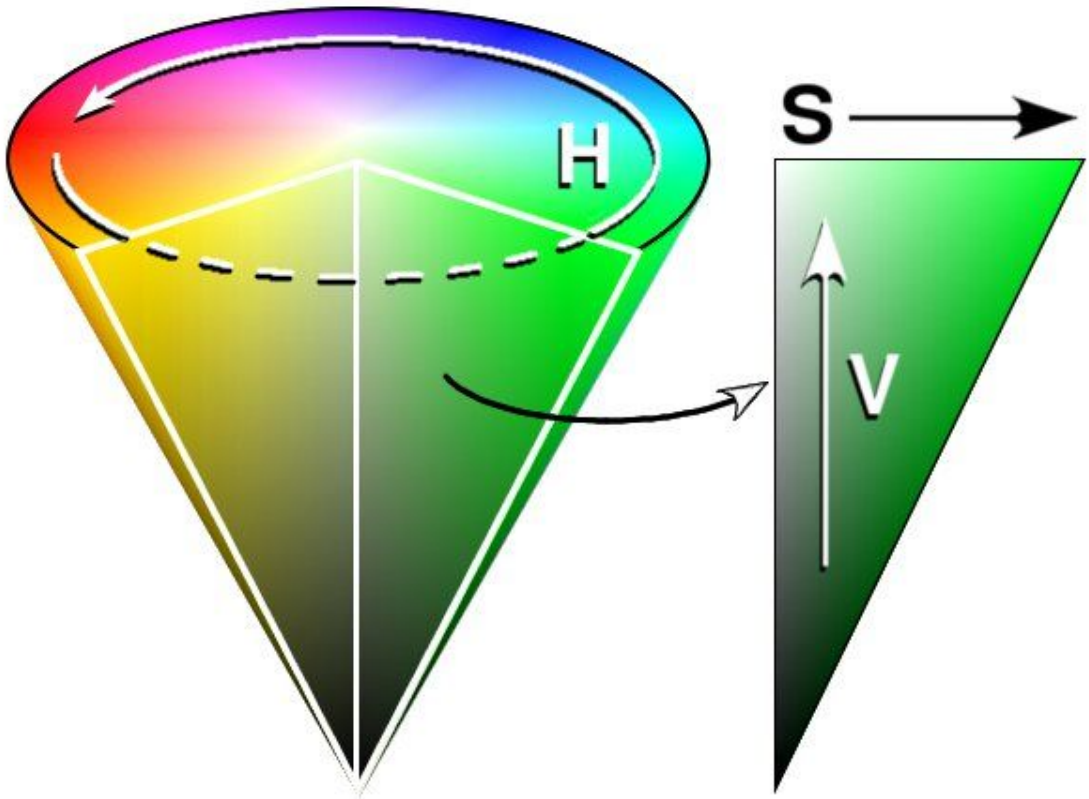
Path tracing:



3.5 Barwa światła, parametry materiałowe, mieszanie barw.

RGB
CMYK - połączenie wszystkich daje czerń (Cyan / Magenta / Yellow / Black)
HSV (Hue / Saturation / Value (jasność))



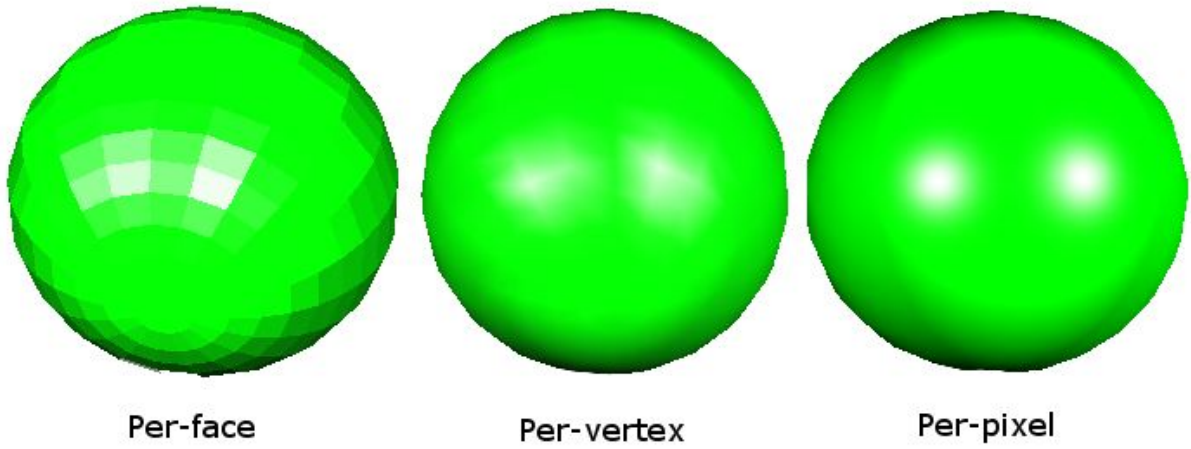


3.6 Oświetlenie liczone per vertex i per pixel – czym się różnią?

(Niech mnie ktoś poprawi jeśli się mylę) Oświetlenie per pixel jest liczone z punktu widzenia piksela (sprawdzamy jakie światło pada na ten piksel), przez co kolejne klatki nie mogą korzystać z obliczeń wcześniejszych, jeśli zmieniamy ustawienie kamery. Przy obliczaniu oświetlenia per vertex liczymy oświetlenie dla wierzchołka i obracanie kamery nie ma

żadnego wpływu na oświetlenie tegoż wierzchołka, więc możemy korzystać z poprzednich obliczeń przy rysowaniu kolejnych klatek.

A moze chodzi o to, ze gdy policzymy per vertex to mozemy sobie raptem dodac wartosci z wierzcholkow i policzyc srednia, i w ten sposob oswietlic, a per pixel - te dwie rozne interpolacje.



http://stackoverflow.com/questions/19292893/per-vertex-shading-vs-per-fragment-shading-on-large-models

3.7 Interpolowanie oświetlenia.

Cel: osiągnięcie ciągłego oświetlenia - gładkich przejść.
Gouraud, Phong

3.8 Model Phonga oświetlenia połyskliwego

Oświetlenia Phonga - model oświetlenia stosowany w grafice komputerowej służący do modelowania odbić zwierciadlanych od nieidealnych obiektów. Model ten nie ma podstaw fizycznych, ale dobrze przybliża charakterystykę powierzchni dla których został stworzony.

Model ten przyjmuje, że powierzchnia obiektu jest pokryta bardzo cienką przezroczystą warstwą, na której zachodzi odbicie zwierciadlane, tzn. światło nie zmienia swojej barwy, natomiast na powierzchni znajdującej się pod tą warstwą następuje odbicie rozproszone, które zabarwia światło na kolor przypisany do obiektu. W świecie rzeczywistym takimi właściwościami cechują się np. błyszczące plastiki czy powierzchnie pomalowane bezbarwnym lakierem.

3.9 Idea metody śledzenia promieni

W tej metodzie tak naprawdę sprawdzamy skąd przyszły promienie padające na dany piksel. Przez każdy piksel przepuszczamy foton (ewentualnie kilka) i śledzimy drogę tego promienia, uwzględniając odbicia i przenikanie przez powierzchnie przezroczyste. W ten sposób obliczamy kolor światła jakie pada na dany piksel.

3.10 Idea metody energetycznej (radiosity)

Każda powierzchnia pochłania światło, ale także odbija jego część. Wyznaczanie globalnego rozkładu natężenia światła tą metodą uwzględnia powyższe zjawisko.
Radiosity uwzględnia **jedynie odbicia rozproszone**, stąd:

- Intensywność światła niezależna od kierunku
- Brak efektów świetlnych związanych z położeniem obserwatora (rozbłyski na powierzchniach metalicznych, odbicia lustrzane, załamanie światła itp.)

W połączeniu z metodą śledzenia promieni pozwala na uzyskanie dobrych efektów wizualnych (ray tracing uwzględnia efekty świetlne związane z położeniem obserwatora)
https://pl.wikipedia.org/wiki/Metoda_energetyczna

3.11 Zasada nakładania tekstur.

Jeden z kluczowych problemów – jak mapować teksturę 2d na obiekt 3d
W ogólności stosujemy przekształcenie typu: x = X(s,t); y = Y (s,t); z = Z (s,t);
UV Mapping :
1.Rozwinięcie siatki modelu
2.Nałożenie tekstury na siatkę
3. Zwinięcie siatki z teksturą

Rodzaje mapowania: planar, cubic, cylindrical, spherical

3.12 Tekstury proceduralne

Tekstura proceduralna – tekstura tworzona na podstawie określonych procedur matematycznych (algorytmów). Tekstury proceduralne charakteryzuje praktycznie nieskończona dokładność, bowiem w odróżnieniu od tekstur bitmapowych, kolor punktu jest funkcją współrzędnych rzeczywistych, a nie całkowitych. Możliwe jest więc dowolne powiększanie takiej tekstury – na tyle, na ile pozwala precyzja obliczeń.

Przykłady tekstur proceduralnych:

- szachownica,
- wzór typu plaster miodu,

- różne gradienty,
- chmury,
- marmur,
- drewno,
- szum perlina (2 i 3 wymiarowy).

Ważną cechą tekstur proceduralnych jest możliwość animacji ich parametrów (np. kolorów).

3.13 Mipmapping

Mipmapping to technika teksturowania bitmapami wykorzystywana w grafice trójwymiarowej, która pozwala uniknąć artefaktów i tym samym uzyskać lepszą jakość obrazów. Przyspiesza ona również sam proces teksturowania.

Mipmapping wykorzystuje serię *MIP map* (*mipmap*), tj. tekstur o różnych rozmiarach, które są wynikiem skalowania wyjściowej tekstury. Jeśli wyjściowa tekstura ma rozmiar będący potęgą dwójki $2^n \times 2^n$, to zostanie wygenerowanych n *mipmap* o rozmiarach $2^{n-1} \times 2^{n-1}$, $2^{n-2} \times 2^{n-2}$, itd., aż do tekstury o rozmiarach 1x1 pikseli. Np. jeśli $n = 8$, to kolejne *mipmapy* mają rozmiary: 256x256 (oryginalna tekstura), 128x128, 64x64, 32x32, 16x16, 8x8, 4x4, 2x2 i 1x1.

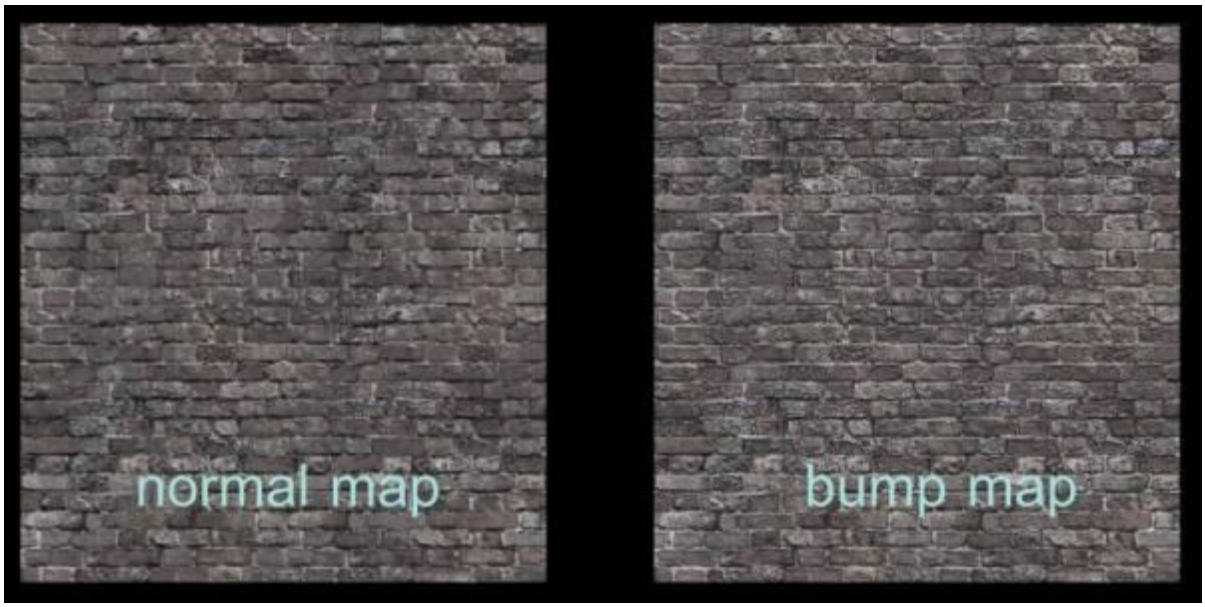
Skalowanie może być wykonane w prosty sposób poprzez uśrednianie pikseli, albo jedną z bardziej zaawansowanych technik filtrowania. Kolejne *mipmapy* są wstępnie pozbawione zakłóceń.

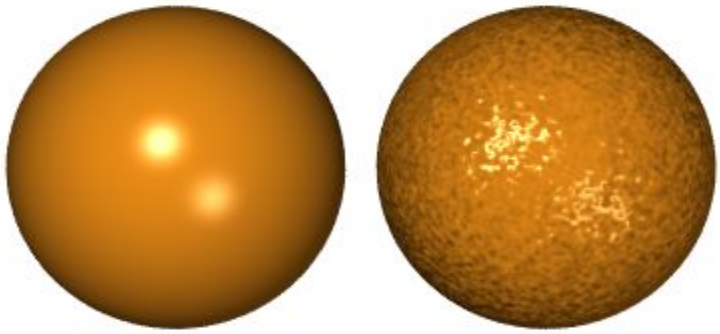
Następnie do poteksturowania obiektu trójwymiarowego używa się tej tekstury, której rozdzielczość jest wystarczająca do reprezentowania obiektu obserwowanego z pewnej odległości i to właśnie od odległości zależy, która *mipmapa* zostanie wybrana. Im obiekt znajduje się dalej od obserwatora, tym mniejszą zajmuje powierzchnię i tym mniejsza tekstura jest potrzebna. Np. jeśli obiekt ma wymiary ok. 10x13 pikseli, to do jego poteksturowania wystarczy tekstura 16x16, jeśli obiekt ma ok. 54x40 to wystarcza tekstura 64x64 itd. W obu przykładach nie ma sensu odwoływać się do tekstury o najwyższej rozdzielczości (powiedzmy 256x256) bo i tak duża część pikseli nigdy nie będzie widoczna.

W praktyce nie jest wybierana jedna *mipmapa*, ale brane są dwie najbliższe i dokonywana jest ich interpolacja.

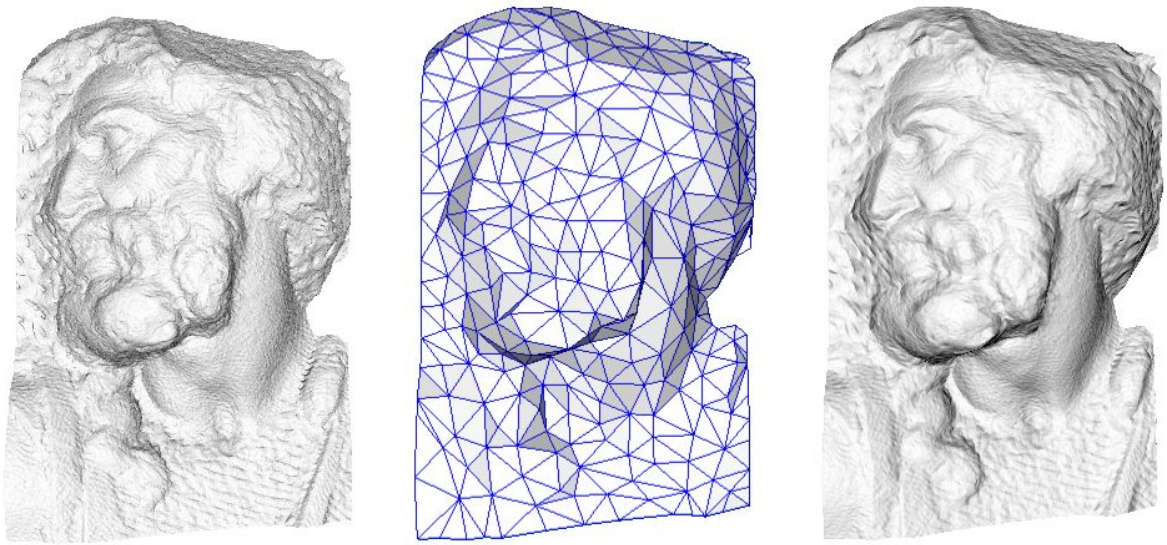
3.14 Efekty bump-mapping, normal mapping i parallax mapping

Bump mapping: **Mapowanie wypukłości** – technika teksturowania, która symuluje niewielkie wypukłości powierzchni, bez ingerencji w geometrię obiektu trójwymiarowego. Technika polega na użyciu tekstury, która nie jest jednak bezpośrednio wyświetlana, ale powoduje lokalne zakłócenia (obrót) wektora normalnego. Ponieważ każdy model oświetlenia (np.oświetlenie Phonga) w jakiś sposób wiąże kąt pomiędzy promieniem światła a wektorem normalnym, to rezultatem zakłóceń jest pojawienie się na obrazie złudzenia nierówności powierzchni. Efekt jest bardzo przekonujący, większość ludzi nie zwraca uwagi na fakt, że brzegi obiektu pozostały "niezakłócone".





Mapowanie normalnych Jest rozwinięciem koncepcji mapowania wypukłości. Podstawą działania mapowania normalnych jest zastąpienie wektorów normalnych opisanych przez geometrię, wektorami zapisanymi w specjalnej teksturze - mapie normalnych. Mapowanie normalnych pozwala w bardzo efektywny sposób, uzyskać wizualne złudzenie większej złożoności modelu trójwymiarowego niż jest w rzeczywistości. Dzięki temu liczba wielokątów potrzebnych do opisania takiego modelu może być znacznie ograniczona.



original mesh
4M triangles

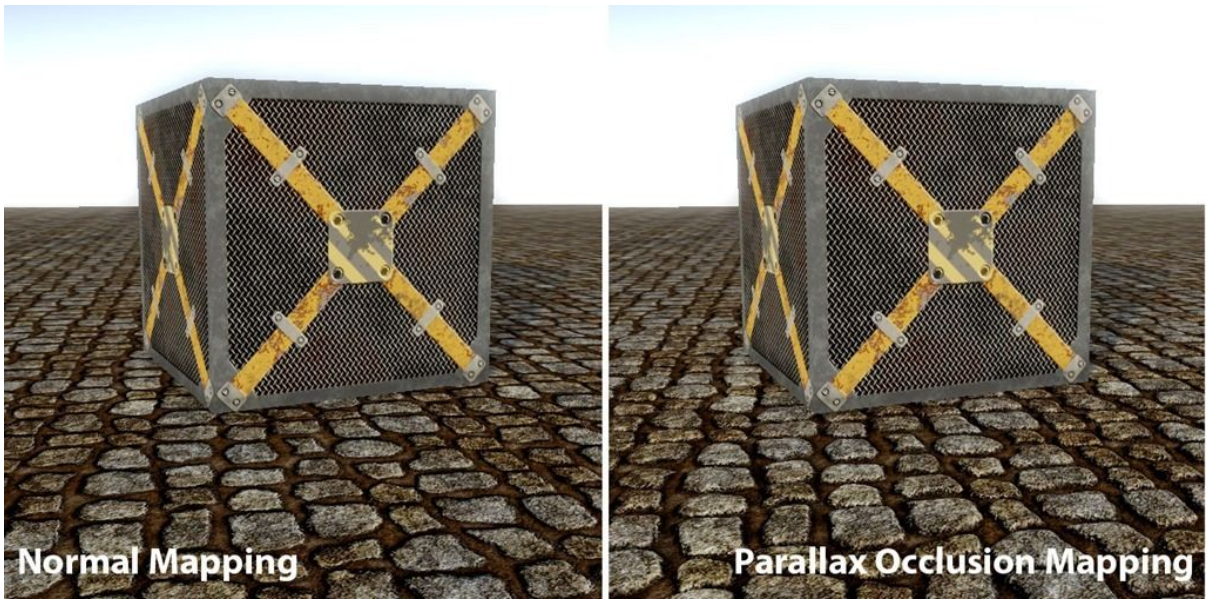
simplified mesh
500 triangles

simplified mesh
and normal mapping
500 triangles

Mapowanie paralaksy jest to technika wizualizacji wybojów na powierzchni obiektów 3D, przez odwzorowanie paralaksy tych wybojów widocznej podczas ich obserwacji pod kątem. Paralaksę odwzorowuje się przez przesunięcie tekstur obiektu.

Mapowanie paralaksy wykorzystuje mapę wysokości, czyli teksturę z zapisanymi wysokościami poszczególnych punktów. Klasyczne mapowanie paralaksy polega na odczytaniu wysokości h w punkcie tekstury, po czym znajduje się przesunięcie.

Ponieważ ta technika zakłada, że wysokość nie zmieni się od punktu obserwowanego do punktu widocznego (wynikającego z przesunięcia), daje ona poprawne efekty tylko gdy rozmiary wybojów są wystarczająco duże. Małe, a wysokie wyboje będą generować zakłócenia obrazu. W tej technice nie widać też zasłaniania jednych wybojów przez drugie i nie jest możliwe rzucanie przez nie cieni, dlatego cieniowanie uzyskuje się poprzez mapowanie normalnych.



3.15 Zastosowania tekstur 2D i 3D

- Tekstury 3D - efekty wolumetryczne - ogień, mgła, dym, może przedstawiać światło

4 Elementy modelowania

4.1 Modelowanie za pomocą funkcji parametrycznych.

4.2 Cechy wielomianu Beziera i Hermita

Wielomian Beziera (czy tu chodzi po prostu o krzywe?) - krótki artykuł => http://www.zobaczycmatematyke.krk.pl/025-Zolkos-Krakow/bezier.html

Cechy (http://informatyka.wroc.pl/node/14):
Krzywe Béziera mają wiele ciekawych własności. Krzywa $\mathbf{P}$ *zaczyna się* w punkcie $\mathbf{W}_0$, bo $\mathbf{P}(0) = \mathbf{W}_0$, a *kończy* — w punkcie $\mathbf{W}_n$, bo $\mathbf{P}(1) = \mathbf{W}_n$. Cała krzywa znajduje się w *otoczce wypukłej* (ang. *convex hull*) swoich punktów kontrolnych, tj. w najmniejszym wielokącie wypukłym zawierającym punkty $\mathbf{W}_0$, $\mathbf{W}_1, \ldots, \mathbf{W}_n$, co można zaobserwować na animacji 2. (zobacz też zadanie 3.). Własność ta ma duże znaczenie praktyczne. Pozwala ona bowiem stwierdzić w jakim obszarze znajduje się wykres krzywej. Wykorzystuje się to np. przy sterowaniu robotami pracującymi w halach fabrycznych. Ruch ich ramion jest często opisany właśnie krzywymi Béziera. Wykorzystując własność otoczki wypukłej można roboty zaprogramować tak, aby mieć pewność, że nie dojdzie do kolizji ich ramion.

Wielomiany Hermite'a (Z MOWNiTu :P) - **Wielomiany Hermite'a** – wielomiany o współczynnikach rzeczywistych, będące rozwiązaniem równania rekurencyjnego

$$H_{n+1}(x) = 2xH_n(x) - 2nH_{n-1}(x),$$

przy warunkach początkowych

$$H_0(x) = 1$$
$$H_1(x) = 2x$$

Własności:

- $H_n(x)$ jest wielomianem n-tego stopnia.
- $\frac{dH_n(x)}{dx} = 2nH_{n-1}(x)$
- $H_{2n}(0) = (-1)^n \frac{(2n)!}{n!}$
- $H_n(-x) = (-1)^n H_n(x),$

czyli dla n parzystego $H_n(x)$ jest funkcją parzystą, a dla n nieparzystego - funkcją nieparzystą.

- $\int_{-\infty}^{\infty} H_n(x)H_m(x)e^{-x^2}dx = \sqrt{\pi}2^n n!\delta_{nm},$

czyli wielomiany Hermite'a tworzą układ wielomianów ortogonalnych z funkcją wagową e^{-x^2}.

4.3 Modelowanie za pomocą funkcji uwikłanych

http://www.orefind.com/blog/orefind_blog/2014/11/24/what-is-implicit-modelling-part-1

4.4 Porównanie podejść z funkcjami parametrycznymi i uwikłanymi