dokumentów23. Parametr reprezentacja krzywych i powierzchni. Jaka jest interpre macierzy ograniczeń geometrycz G i macierzy bazowej M w paramet przedst krzywych Hermite'a i Bezier?

- Tego jeszcze nie było na wykładzie jak coś
- Ok, a dokąd doszły wykłady?

Niewiele dalej niż ostatni wrzucony przez Alde wykład. Omówił jeszcze metodę energetyczną i chyba przymierzał się do tekstur.
Na ostatnim wykładnie były fraktale, generowanie losowych scen itp.

Zagadnienia - link
Materiały wszelakie od Aldy

I. ELEMENTY OPENGL

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

Tak - **GL_DEPTH_TEST** jest nim.
http://www.arcsynthesis.org/gltut/Positioning/Tut05%20Overlap%20and%20Depth%20Buffering.html

2. Czego dotyczy w OpenGL podwójne buforowanie obrazu?

Mamy 2 bufory na obraz: obraz jednego bufora jest wyświetlany aktualnie na ekranie, a drugi służy do rysowania. Po skończeniu rysowania, podmieniamy bufory. Dzięki temu unikamy rysowania bezpośrednio na ekranie (użytkownik tego nie widzi) i uzyskujemy płynniejszą animację (+ redukcja/eliminacja artefaktów). Wymaga 2x więcej pamięci potrzebnej przy określonej rozdzielczości.

3. Do czego służy bufor VBO?

Vertex Buffer Object jest buforem służącym jako źródło macierzy wierzchołków dla Vertex Array Object. Bufor ten przekazywany jest do karty graficznej w całości, przyspiesza to zarówno przesył danych (wiele wierzchołków na raz) jak i poprawia wydajność w operowaniu na nich przez vertex shadery. (Oprócz wierzchołków możemy w ten sposób przekazywać również m. in.: wektory normalne, mapowanie tekstur, kolory)

W biblii trochę inaczej:

“A VBO is a buffer object that represents storage for vertex data. Data can be placed in these buffers with hints that tell OpenGL how you plan to use it, and OpenGL can then use those hints to decide what it will do with that data.” A niżej opisane że VAO to tylko kontener na VBO

4. Do czego służą zmienne typu Uniform?

Opisują 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 shader’ach.

5. Nowe typy w GLSL

- double i macierze double
- ?
- http://www.opengl.org/documentation/glsl/ - strona 6 każdego cheatsheeta
- Chyba, że chodzi o nowe w stosunku do języków programowania:
- wektorowe - vec2, vec3, vec4 (wektory float); bvec - bool, ivec - integer, uvec - unsigned
- macierzowe - mat2, mat3, mat4 ogólnie: mat{wiersze x kolumny}
  2 <= wiersze,kolumny <=4 //tak lepiej? :)

6. Rodzaje shaderów

W kolejności przetwarzania:
Vertex shader - wykonuje następujące transformacje (operacje na wierzchołkach):

- położenia (pomocne są tu macierze widoku i rzutowania)
- normalnych (wraz z ich normalizacją)
- współrzędnych tekstur

Ponadto oblicza oświetlenie i kolor W WIERZCHOŁKACH.

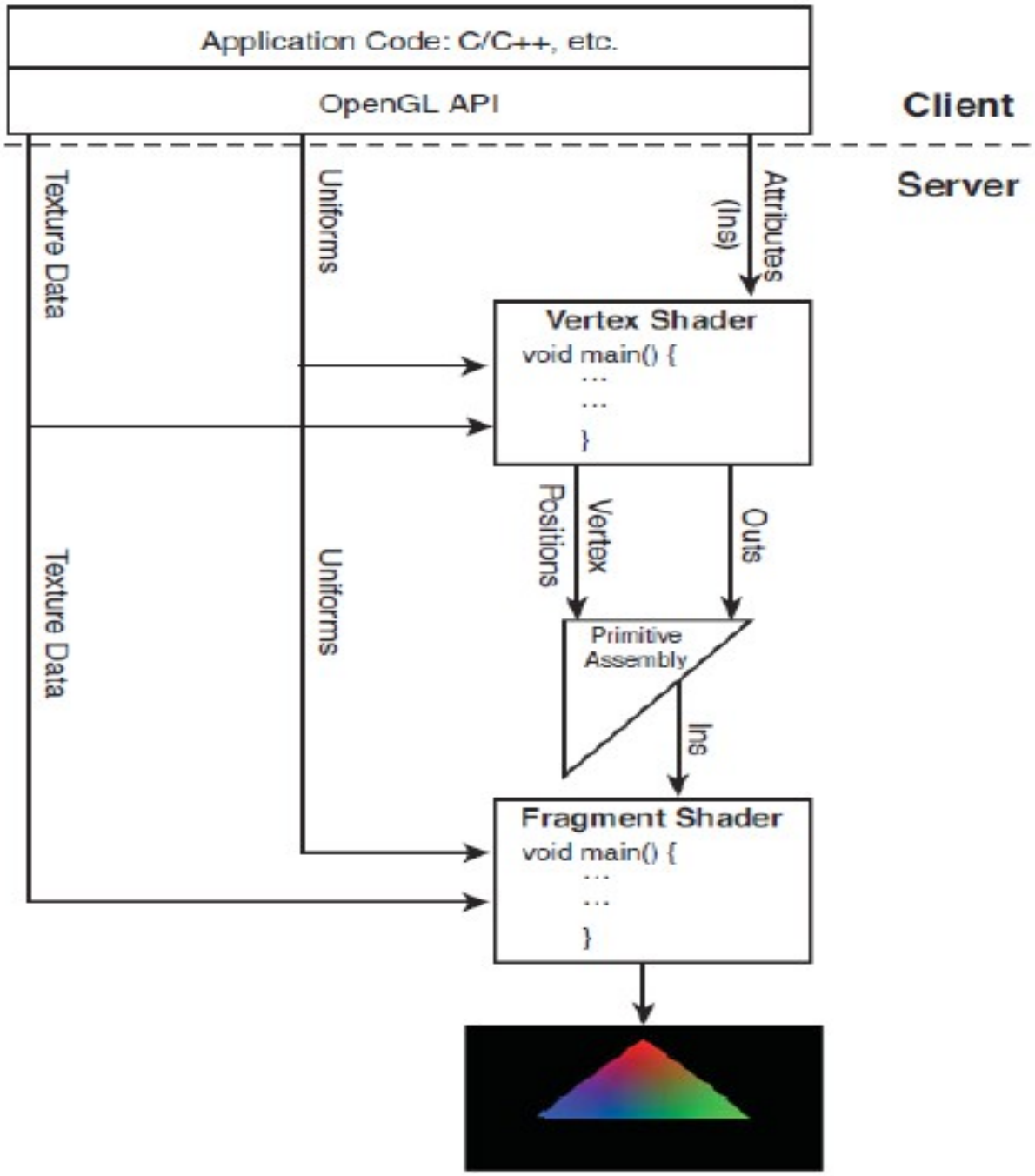
- Wierzchołki przetwarzane są pojedynczo i niezależnie.

Geometry shader - przetwarza GRUPY wierzchołków tworzących podstawowe elementy geometryczne. Może dodawać i kasować wierzchołki oraz zmieniać typy i atrybuty. Geometry shader jest opcjonalny.
3
Fragment (pixel) shader - działa na fragmentach (pikselach) - obrazie powstałym na skutek rasteryzacji. Operacje:

- obliczanie kolorów i współrzędnych tekstur dla każdego piksela
- obliczanie mgły
- obliczanie normalnych oświetlenia dla każdego piksela.

Nie może on zmienić współrzędnych piksela.

7. Schemat potoku graficznego



II. TRANSFORMACJE

1. Jak reprezentowane są transformacje?

Macierze

2. Dlaczego macierze 4x4?

Współrzędne jednorodne mają N+1 współrzędnych dla N wymiarowej przestrzeni.
Dlaczego?

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, 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
Tak dodatkowo - jak mamy macierz 4x4 to można wszystkie przekształcenia zapisać jedną macierzą.

3. Podstawowe macierze transformacji

1. Translation (x, y, z [, w?])

$$\mathbf{T} = \begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

2. Rotation (angle, x, y, z)

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

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

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

3. Scaling (x, y, z)

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

4. Kolejność składania transformacji.

Kolejność transformacji ma znaczenie! ⇔ Mnożenie macierzy nie jest przemienne

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://www.theamazingking.com/ogl-matrix.php

6. Rodzaje rzutowania

1. **Rzutowanie ortogonalne**
   - GLFrustum::SetOrthographic(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)

$$P = \begin{bmatrix} \frac{2}{right-left} & 0 & 0 & \frac{right+left}{right-left} \\ 0 & \frac{2}{top-bottom} & 0 & \frac{top+bottom}{top-bottom} \\ 0 & 0 & \frac{2}{near-far} & \frac{far+near}{far-near} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2. **Rzutowanie perspektywistyczne**
   - GLFrustum::SetPerspective(GLfloat fovy, GLfloat aspect, GLfloat near, GLfloat far)
   - aspect - w/h
   - fovy - kąt widzenia wpłaszczyźnie yz
3. **Viewport**: rzutowanie na fragment okna
   - glViewPort(0, 0, w, h);

III. OŚWIETLENIE

1. Dlaczego w OpenGL 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.
Dlatego wprowadzone zostało oświetlenie *ambient*, oświetlające wszystko(raczej: cokolwiek), a w szczególności obiekty do których nie dociera źródło światła (inaczej, byłyby niewidoczne - czarne). Oświetlenia *diffuse* potrzebujemy do światła rozproszonego (mające źródło), które jest odbijane od powierzchni równomiernie, ale w zależności od kąta pod jakim na nią pada.
Specular jest podobny do *diffuse*, z tą różnicą, że światło nie jest odbijane równomiernie - odbłyski mają kierunek i są odbijane ostro.

2. Znaczenie składowych ambient, diffuse i specular.

- **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 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. 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 potraktorać jako wektor normalny do tej płaszczyzny.
Wektor normalny określa również **stronę** powierzchni. *Co z nią?*

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.

4. Lokalne, a globalne modele oświetlenia.

Oświetlenie lokalne: obiekty traktowane są jako niezależne, nie uzględniamy 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?
za wiki http://pl.wikipedia.org/wiki/O%C5%9Bwietlenie_globalne
modele :

- mapowanie fotonowe,
- path tracing,
- radiosity,

5. Barwa światła, parametry materiałowe, mieszanie barw.

Z trzech barw podstawowych mamy wszystkie barwy. (04_modele_oswietlenia_v1.pdf więcej już ciężko z tych slajdów wyciągnąć...)
Tutaj

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.

7. Interpolowanie oświetlenia.

W skrócie: różne odcienie dla trójkąta, zależne od kąta padania światła. A tuaj powinniśmy pisać o Phoungu i Gouraudzie?

8. Model Phonga oświetlenia połyskliwego

9. Idea metody śledzenia promieni

W tej metodzie tak na prawdę 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.

10. Idea metody energetycznej

Pytania z poprzednich lat

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

Bufor głębokości
Parametr GLUT: **GLUT_DEPTH** przy inicjalizacji (funkcja *glutInitDisplayMode*(...))
W OpenGL: *glDepthMask*(GL_TRUE)
Przełącznik: GL_DEPTH_TEST

2. Do czego służy bufor głębokości w OpenGL?

Patrz pytanie 1.
Wykorzystywany do wyświetlania obrazów 3D, przechowuje współrzędną Z (głębokość, odległość od obserwatora) dla każdego piksela obrazu. Dzięki temu uzyskuje się poprawny obraz, tzn. taki, w którym obiekty 3D są prezentowane zgodnie z ich wzajemnymi relacjami przesłaniania.

3. W czym nam pomaga funkcja glNormalize()?

Mając wyliczoną normalną, funkcja ta normalizuje ją do jedynki. Bez użycia tej funkcji, może zdarzyć się, iż wyliczony przez nas wektor nie jest jednostkowy (długość != 1). Wektor ten pojawia się we wzorze dotyczącym oświetlenia i im jest on dłuższy, to tym jaśniejsze jest oświetlenie.

4. Co to jest tekstura **parametryczna**?

?! chodzi o teksturę proceduralną? jeżeli tak, to:
Tekstura **proceduralna** to taka, która jest tworzone na podstawie określonych procedur matematycznych (algorytmów). Tekstury proceduralne charakteryzuje praktycznie nieskończona dokładność, bo kolor punktu jest funkcją współrzędnych rzeczywistych, a nie całkowitych. Możliwe jest więc dowolne powiększanie takiej tekstury (na tyle, na ile pozwala precyzja obliczeń). Tekstury proceduralne mogą być dwuwymiarowe - wówczas kolor piksela jest funkcją dwóch zmiennych - albo i nawet trójwymiarowe - gdzie kolor punktu jest funkcją jego współrzędnych przestrzennych.
Przykłady tekstur proceduralnych: szachownica, wzór typu plaster miodu, różne gradienty, chmury korzystamy z szumu perlina, marmur, drewno.
Ważną cechą tekstur proceduralnych jest możliwość animacji ich parametrów (np. kolorów).

5. Jakie są zalety i wady funkcji uwikłanych?

Zalety:

- łatwo opisać: $x^2+y^2+z^2=1$
- łatwo określić po której stronie krzywej/powierzchni
- Wady:łatwo obliczyć normalne ???(to jest wada) dlaczego to jest wada?

- przy łączeniu dwóch krzywych/powierzchni może być trudno określić czy kierunki ich stycznych zgadzają się w punkcie łączenia
- jak modelowac połowę sfery?

6. Interpolacja Gouraud'a (czyt: guro)

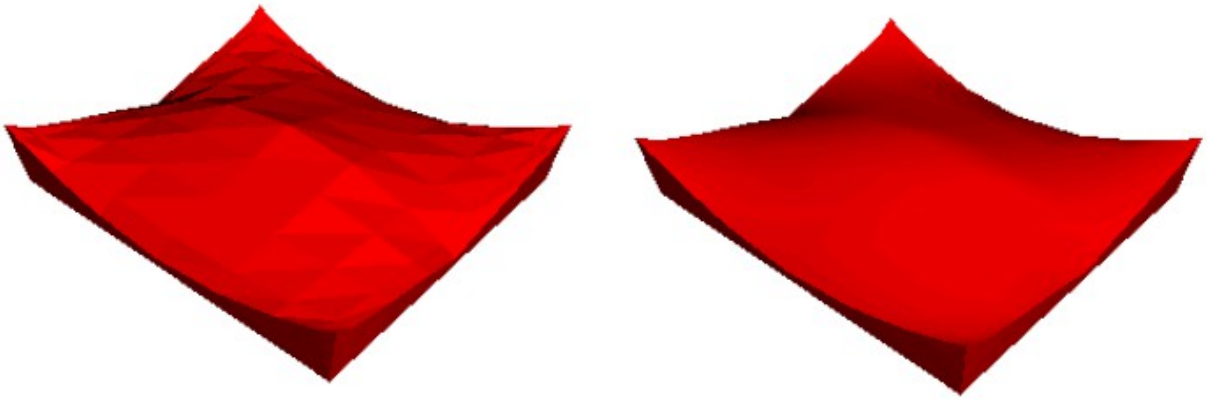
Polega na tym, że staramy się oświetlić pojedynczy wielokąt w różnym stopniu w różnych miejscach. Jest to liniowa interpolacja oświetlenia (jasności i barwy), które zostało wyliczone w wierzchołkach tego wielokąta.

Zalety:

- szybka
- dobrze wygładza krawędzie

Wady:

- wygładza krawędzie również tam, gdzie tego nie chcemy
- nie oblicza odblasków



7. Interpolacja Phonga

Dotyczy nie tylko natężenia światła obliczonego w wierzchołkach, ale również dokonuje interpolacji *normalnych*, które w tych wierzchołkach były zaczepione (opiera się na interpolacji normalnych sąsiadujących wielokątów).

Zalety:

- cieniowanie wysokiej jakości
- odblaski

Wady:

- powolna (kosztowna obliczeniowo)
- jest to metoda przybliżona - czasem jest to widoczne

8. Co oznaczają punkty w opisie krzywych Beziera trzeciego stopnia?

Mamy 4 punkty: P_0, P_1, P_2, P_3.

P_0 - punkt początkowy krzywej

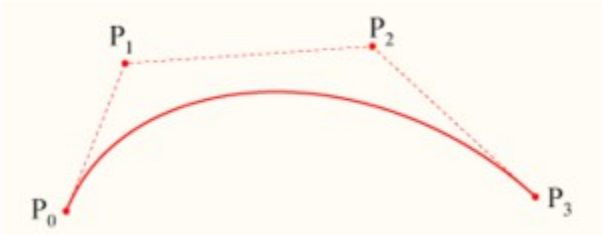
P_3 - punkt końcowy krzywej

Odcinek P_0P_1 jest styczny do krzywej w punkcie P_0

Odcinek P_2P_3 jest styczny do krzywej w punkcie P_3

Początek krzywej jest skierowany w stronę punktu P_1

Koniec krzywej dochodzi do punktu P_3 od strony punktu P_2



9. Co jest najbardziej wymagające obliczeniowo w **ray tracingu**?

Obliczanie punktów przecięcia promienia z obiektem.

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

- ilość obiektów
- kształty obiektów (szybko dla: sfera, płaszczyzna wielokąt; wolno dla skompilokowanych, zbudowanych z wielu wielokątów)
- czy testujemy przecinanie promieni z wszystkimi obiektami na scenie, czy tylko z tymi, z którymi potencjalnie może wystąpić przecięcie

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

Tekstury 3D to funkcje przyjmujące jako parametry położenie punktu w 3 wymiarach i zwracające jego kolor. Wystarczy więc użyć tej funkcji w Shaderze do obliczenia koloru piksela.

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

Są to modele kolorów.
CMYK = **C**yan, **M**agenta, **Y**ellow, Blac**K**

R + B = magenta
R + G = yellow
G + B = cyan

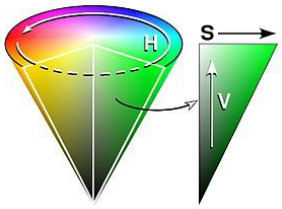
Za pomocą CMY nie można uzyskać koloru czarnego, dlatego powstał CMYK.
CMYK jest wykorzystywany przy drukowaniu.

13. Co to jest HSV?

Model kolorów, nawiązuje do sposob

u w jakim widzi ludzki narząd wzroku.

H = hue
S = saturation
V = value (czasem B - brightness)



14. Jak się liczy normalną i **czym się różni normalna w matematyce od OpenGL-owej?**

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 potraktorać jako wektor normalny do tej płaszczyzny.
Wektor normalny określa również **stronę** powierzchni. *Co z nią?* Standardowo wektory skierowane są na zewnątrz bryły, a podczas rysowania często ustala się żeby rysować tylko zewnętrzną stronę, czyli jeśli kamera znajduje się wewnątrz bryły to powierzchnia bryły niczego nie zasłania bo nie jest rysowana.
I to jest ta różnica - kierunek normalnej ?
Różnica jest taka, że w OpenGL'u normalna jest taka jak sobie ustawisz.
Ano tak - w OpenGL pełna kontrolka nad tematem

@Pytania o kolor

http://wiki.bit.agh.edu.pl/lib/exe/fetch.php?media=studia:przedmioty:grafika:opengl.superbible.5th.edition.pdf
strona 303
odpowiedzi przy założeniu że “<kolor><obiekt>” oznacza że “<obiekt> ma AmbientMaterial=DiffuseMaterial=<kolor>”

15. Jaki kolor będzie miała niebieska kula oświetlana zielonym światłem ambient?
vec3 vAmbientColor = vAmbientMaterial * vAmbientLight
w RGB

Czarny
a nie niebieski ? (http://en.wikipedia.org/wiki/File:Phong_components_version_4.png - tylko tutaj jest niebieskie coś oświetlane białym)

16. Jaki kolor będzie miała żółta kula oświetlona zielonym światłem diffuse
float fDotProduct = max(0.0, dot(vNormal, vLightDir));
vec3 vDiffuseColor = vDiffuseMaterial * vDiffuseLight * fDotProduct;

Odcienie zielonego, przechodzące w czarny tam gdzie jest mały kąt padania światła.

17. Jaki kolor kuli uzyskamy, jeżeli będziemy oświetlać niebieską, czerwoną i żółtą kulę światłem zielonym?
Kula niebieska: czarny
Kula czerwona: czarny
Kula żółta: zielony