gamedesire® academy

22.04.2016

JAK TO DZIAŁA? RZECZ O TWORZENIU SHADERÓW.

Artur Staszczyk

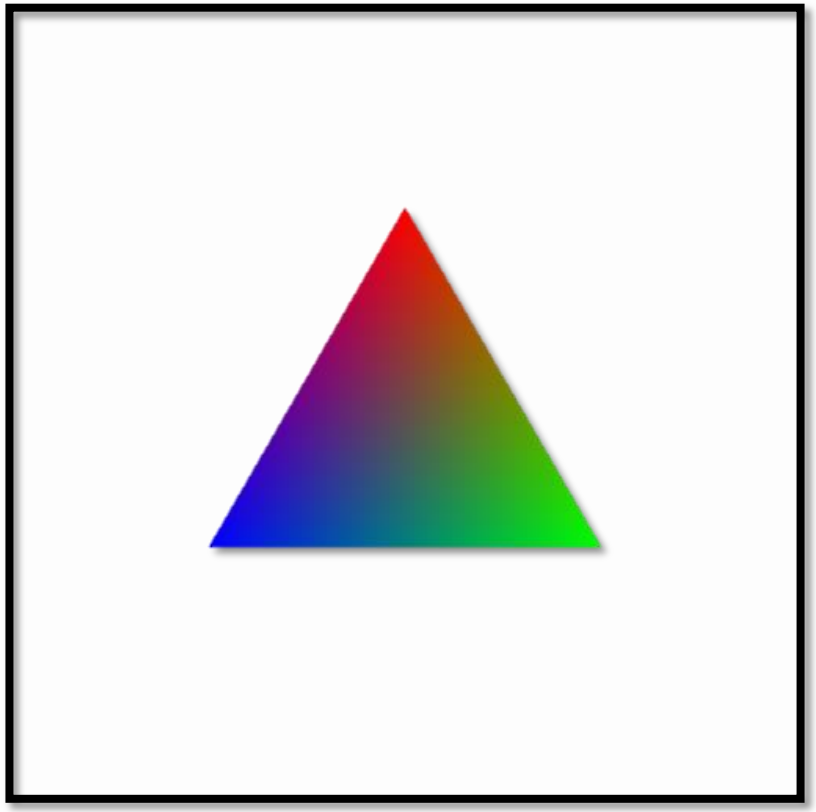
O prezentacji

- Historia (od grafiki na CPU do Unified Shader Model)
- Potok renderowania (wierzchołki, macierze, pipeline, GPU)
- Shadery (vertex + pixel/fragment shaders)
- Przykładowe shadery (Blinn-Phong, Toon Shader, Normal mapping)
- Post-processing
- Unity (jak to działa)

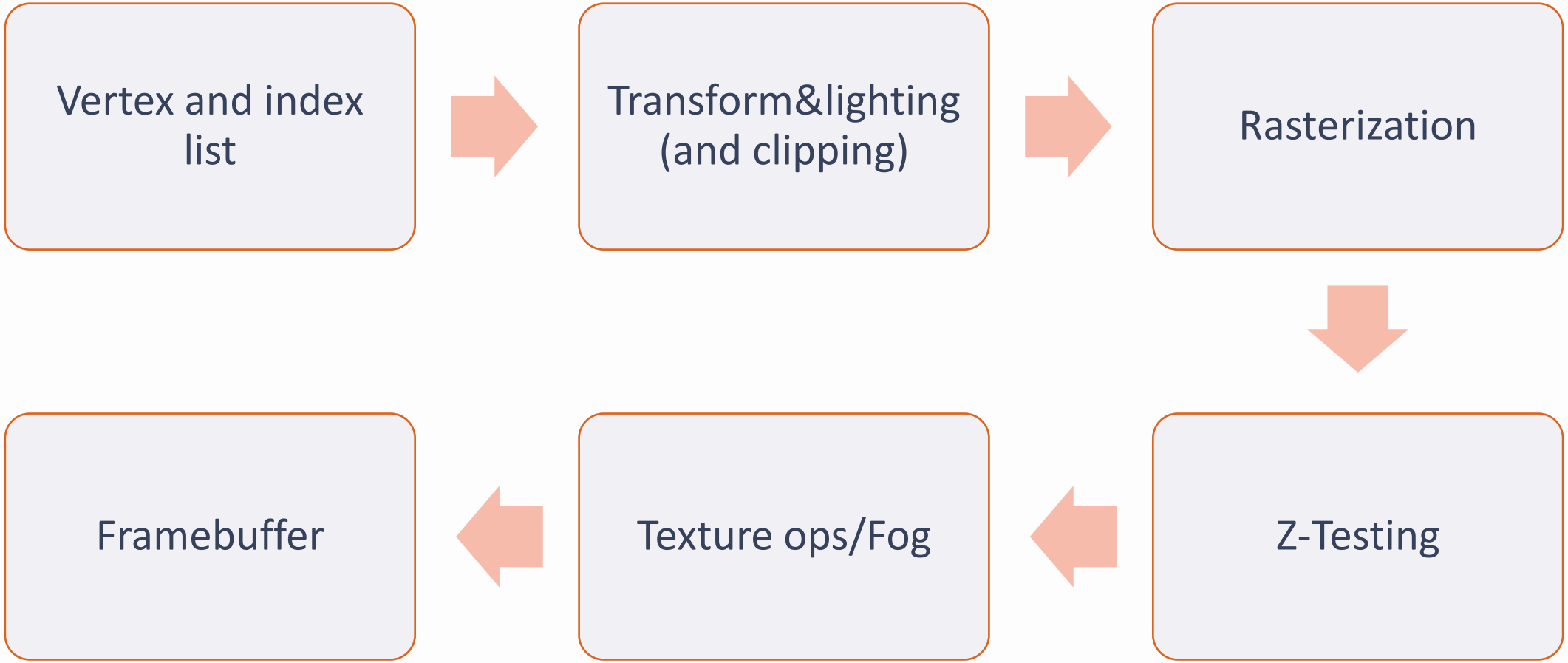
O mnie

- VP Engineering w GameDesire
- W firmie GameDesiere (Ganymede) od 2010 roku
- Wcześniej m.in. CD Projekt RED i Reality Pump
- Przy tworzeniu gier pracuję zawodowo od 9 lat
- Zarówno tytuły indie , AAA jak i narzędzia do tworzenia gier

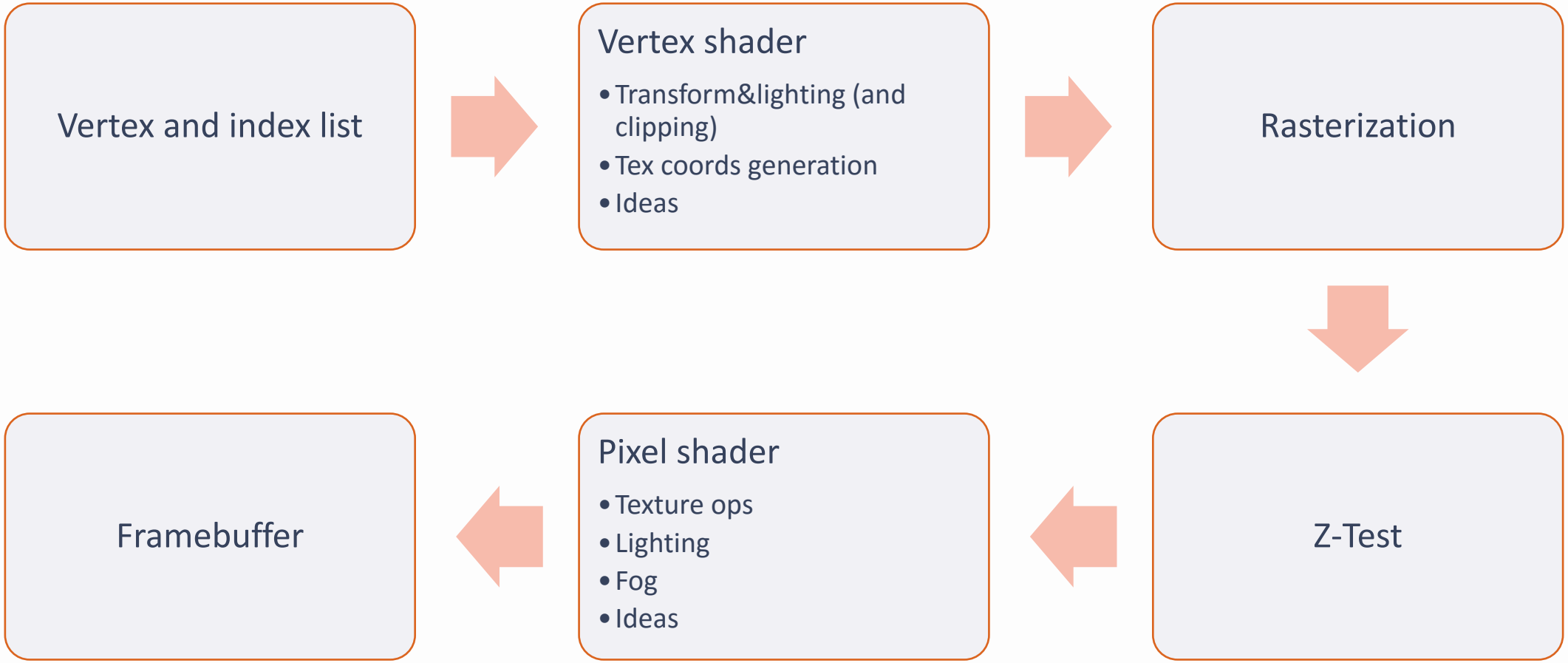
Podstawy



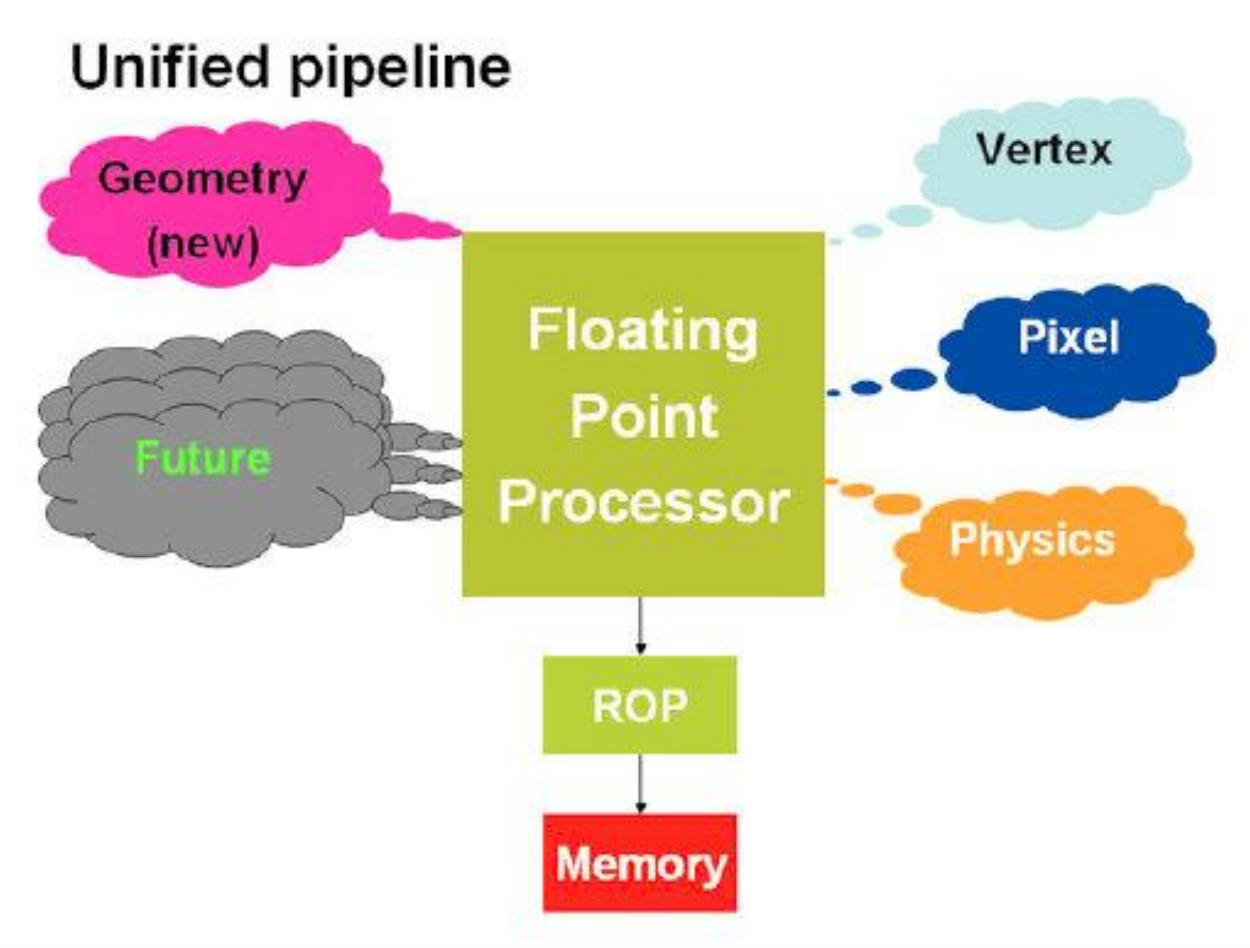
Historia – T&L


Vertex and index list
Transform&lighting (and clipping)
Rasterization
Z-Testing
Texture ops/Fog
Framebuffer

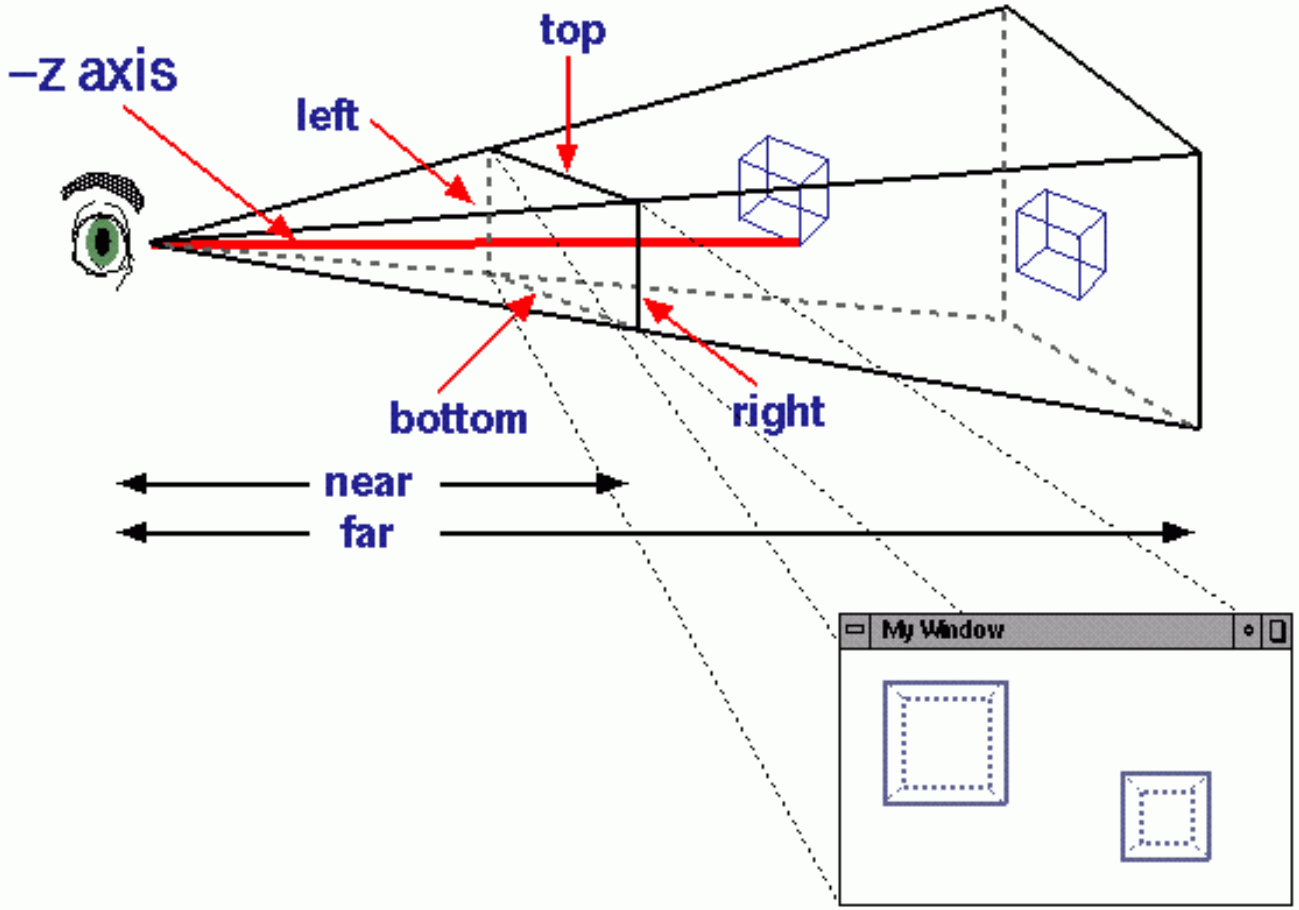
Historia – Shader Model 1.0 – 3.0



Historia – Unified Shader Model



Transformacje

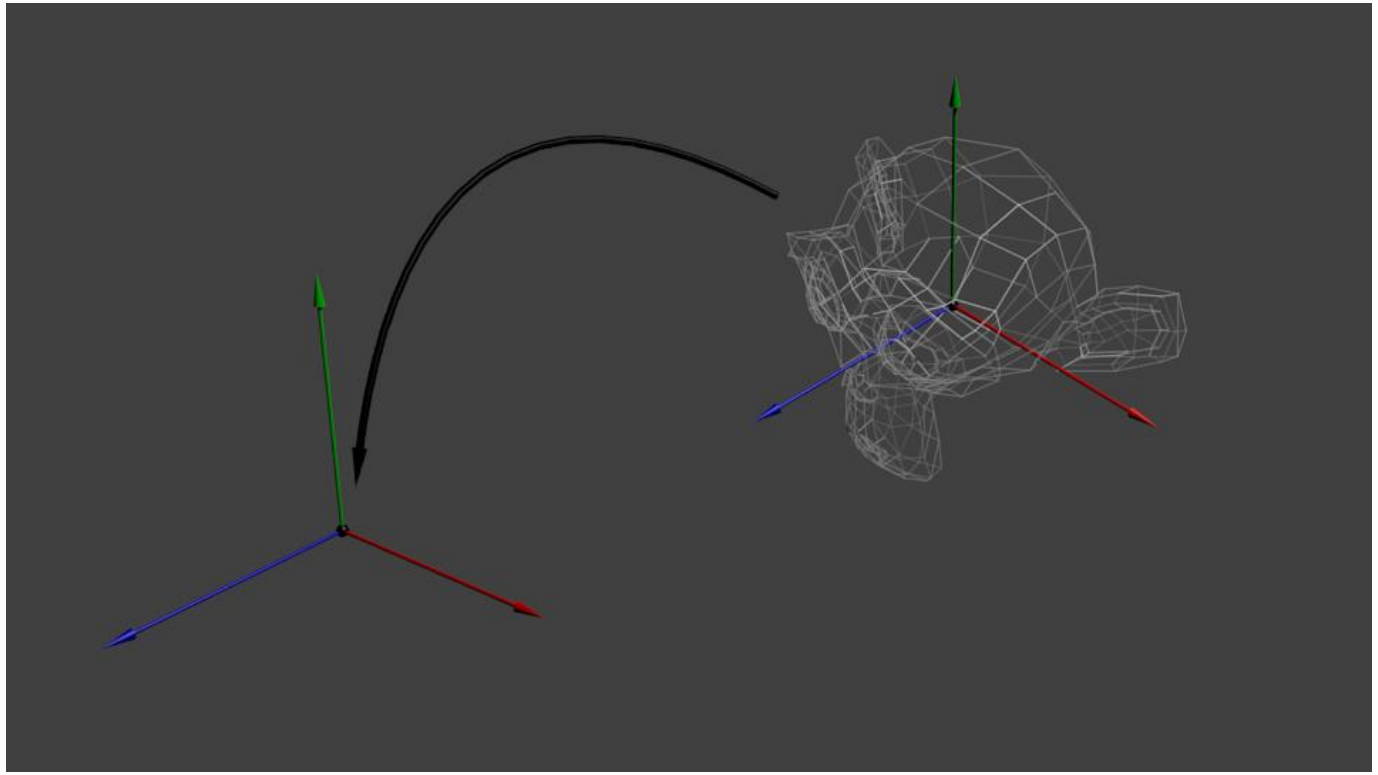


Transformacje

Konwersja sceny 3D na ekran 2D to seria przekształceń macierzowych w przestrzeni wektorowej

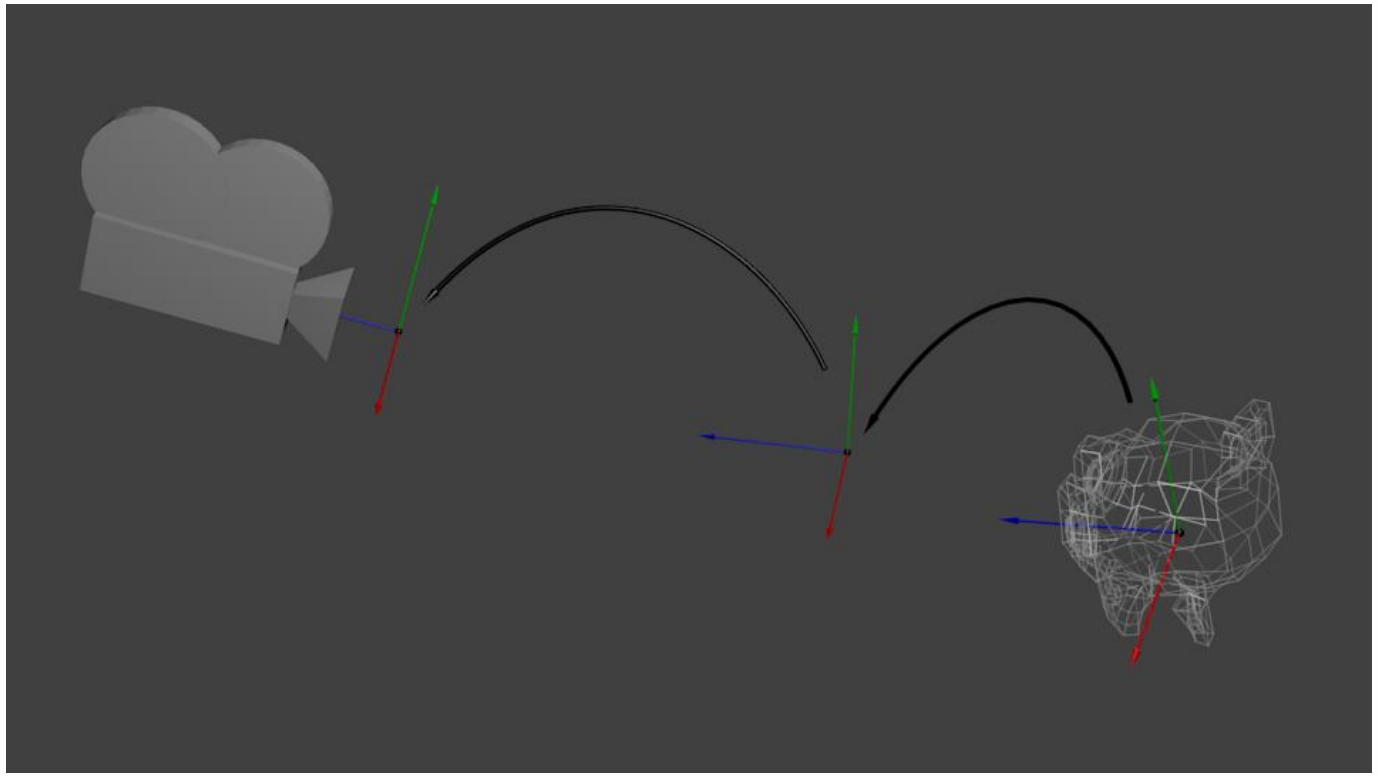
$$\overrightarrow{V'} = \vec{V} * M_{world} * M_{view} * M_{proj}$$

Model/world matrix



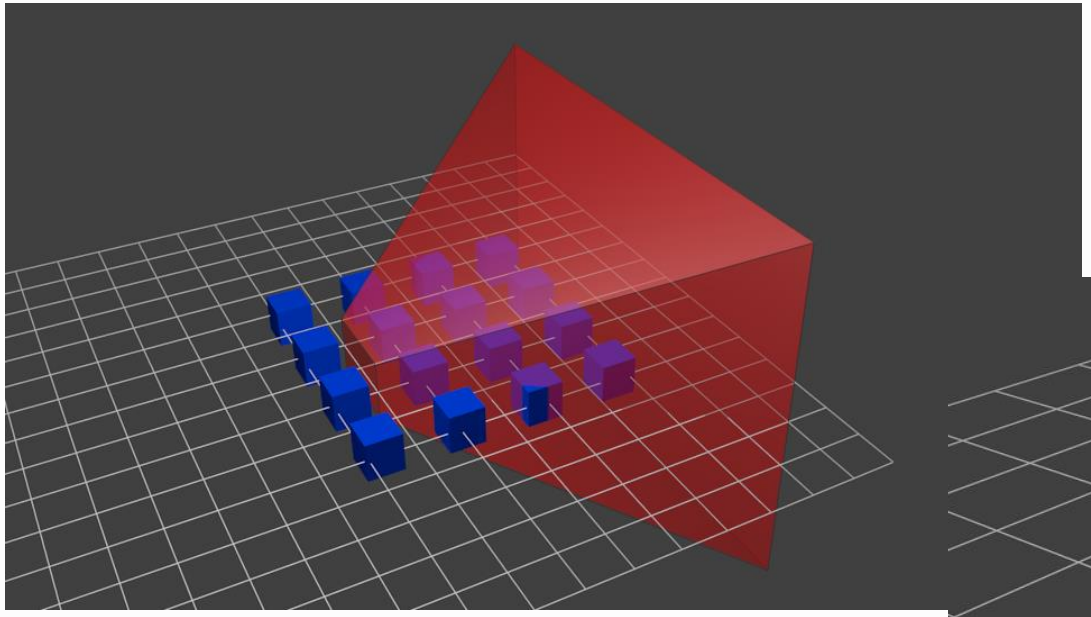
Source: http://www.opengl-tutorial.org/

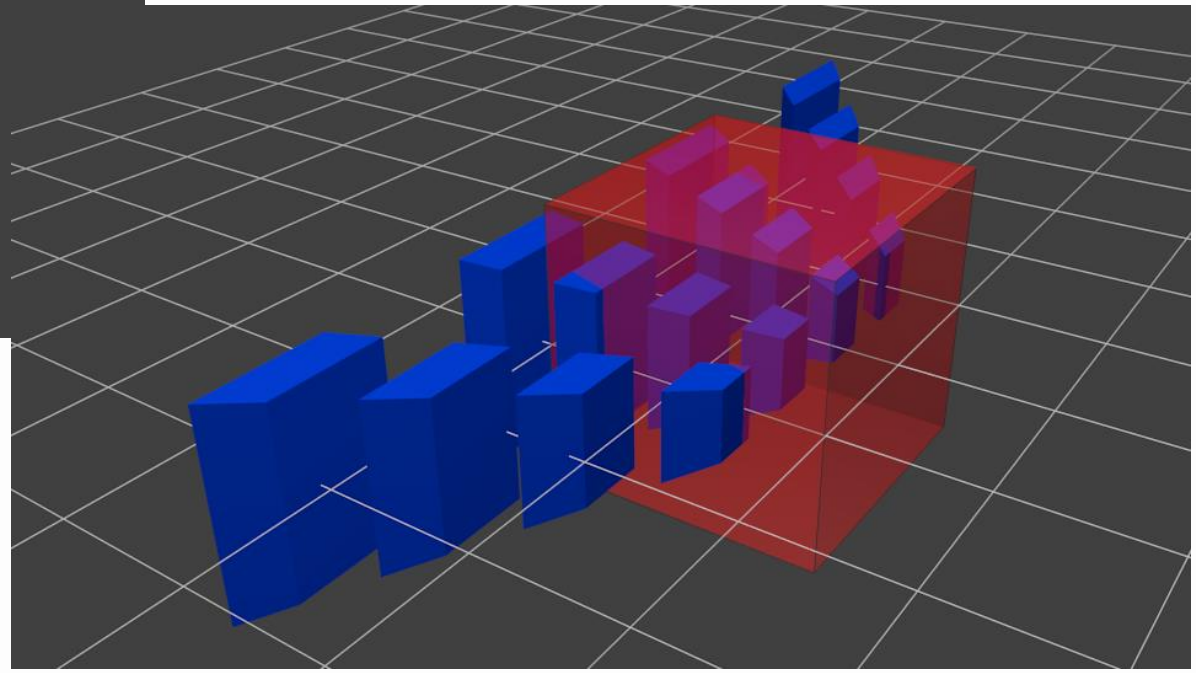
Camera/view matrix



Source: http://www.opengl-tutorial.org/

Projection matrix





Source: http://www.opengl-tutorial.org/

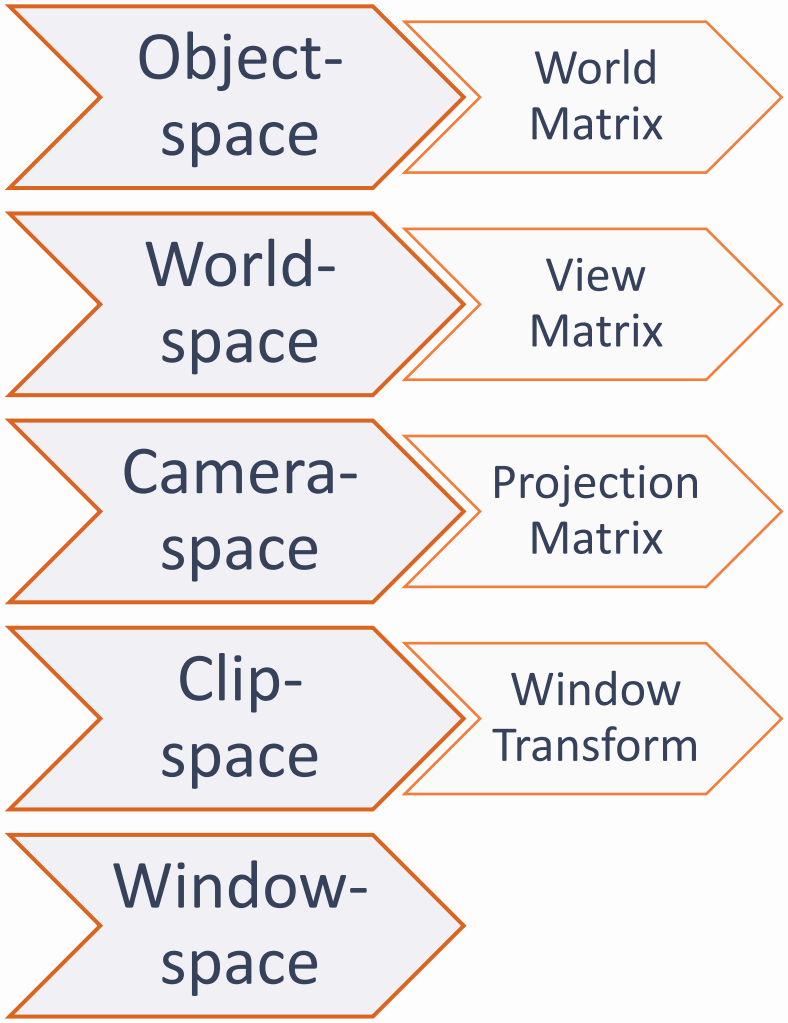
Transformacje

Konwersja sceny 3D na ekran 2D to seria przekształceń macierzowych w przestrzeni wektorowej

$$V' = V * M_{world} * M_{view} * M_{proj}$$

“All problems in computer graphics can be solved with a matrix inversion.”

Jim Blinn



Vertex shader

Uruchamiany jeden raz dla każdego przesłanego wierzchołka

Jego zadaniem jest przeliczyć potrzebne atrybuty (np. kierunek światła, kolor, koordynaty tekstury, pozyzję) i przekazać je do pixel shadera

Pixel shader

Uruchamiany raz dla każdego zrasteryzowanego punktu w celu obliczenia jego koloru

Dla jednego piksela ekranu może być uruchamiany wiele razy (np. dla przezroczystych obiektów)

GPU

Jak GPU przelicza miliony wierzchołków/pikseli w trakcie trwania jednej klatki?

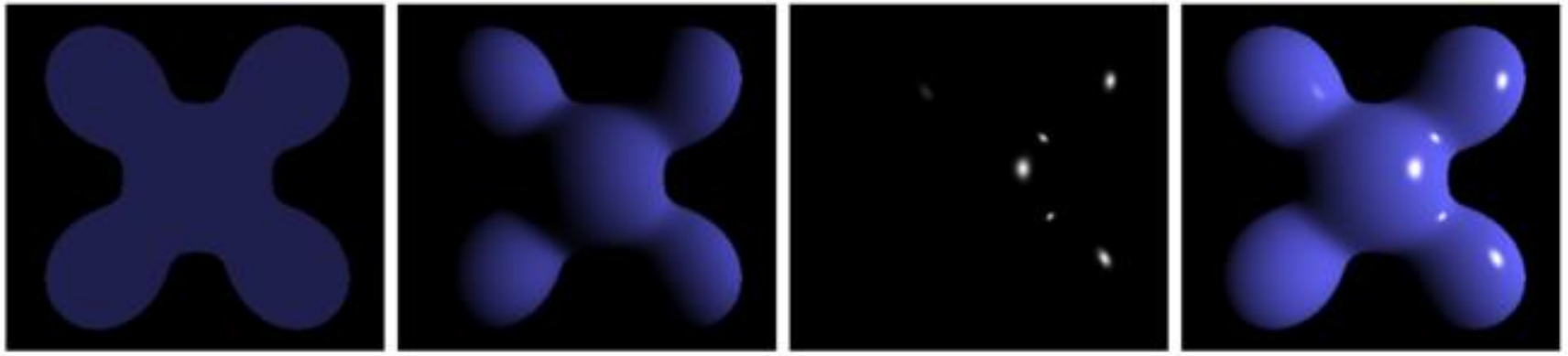
- SIMD (single instruction multiple data)
- Bardzo “lekkie” wątki (brak problemu schedulingu, cache’owania)
- Niezależność między wątkami – brak synchronizacji
- Brak zapotrzebowania na dużą pamięć cache

gamedesire® academy

Przykład

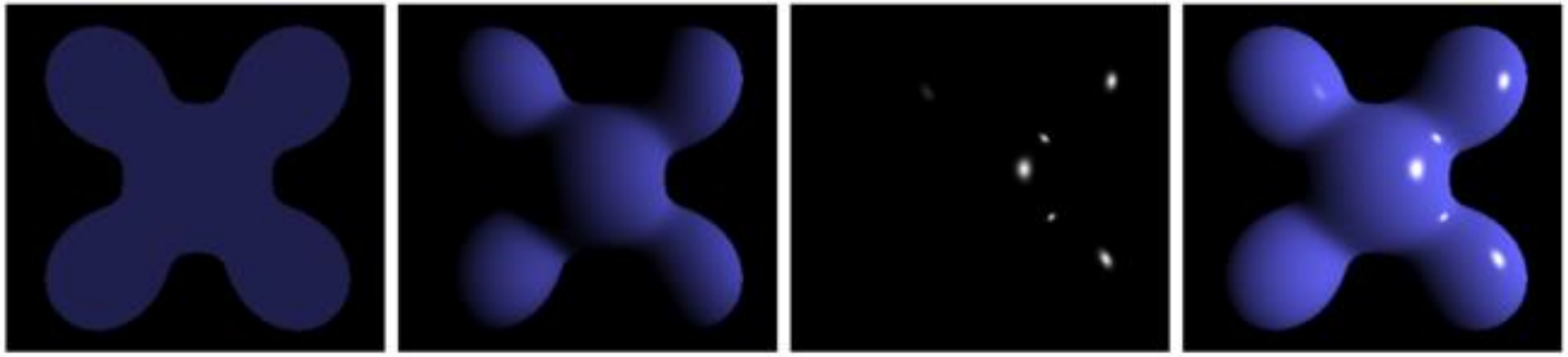
Podstawowy model oświetlenia: Blinn-Phong

- Empiryczny
- Zakłada idealne cechy materiału
- Implementowany przy materiałach dających połysk
- Jest szybki w implementacji
- Jest szybki obliczeniowo



Podstawowy model oświetlenia: Blinn-Phong

$$I_p = k_a i_a + \sum_{lights} (k_d(\vec{L} * \vec{N}) i_d + k_s(\vec{R} * \vec{V})^\gamma i_s$$



Podstawowy model oświetlenia: Blinn-Phong

$$I_p = k_a i_a + \sum_{lights} (k_d(\vec{L} * \vec{N}) i_d + k_s(\vec{R} * \vec{V})^{\gamma} i_s$$

k_a – light ambiet color

k_d – light diffuse color

k_s – light specular color

i_a – material ambient color

i_d – material diffuse color

i_s – material specular color

α – współczynnik połysku

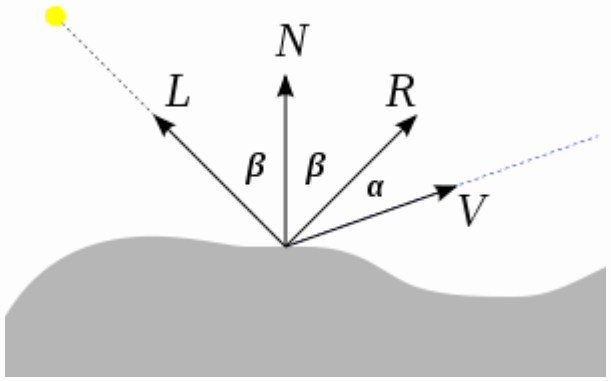
$\vec{L}$ – wektor światła

$\vec{N}$ – normalna punktu

$\vec{R}$ – odbity wektor światła (***reflect(-L, N)***)

$\vec{V}$ – wektor kamery

Wszystkie wektory są **znormalizowane**!!!



Podstawowy model oświetlenia: Blinn-Phong

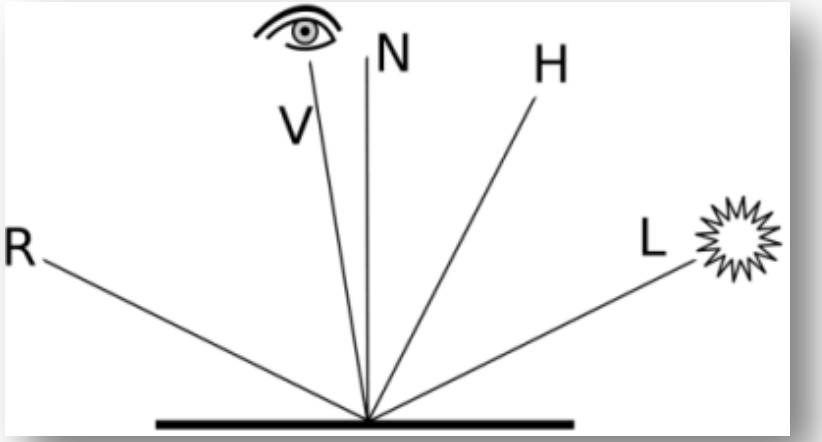
Jakich informacji potrzebujemy?

Per-object: materiał (kolor ambient, diffuse i specular), pozycja kamery

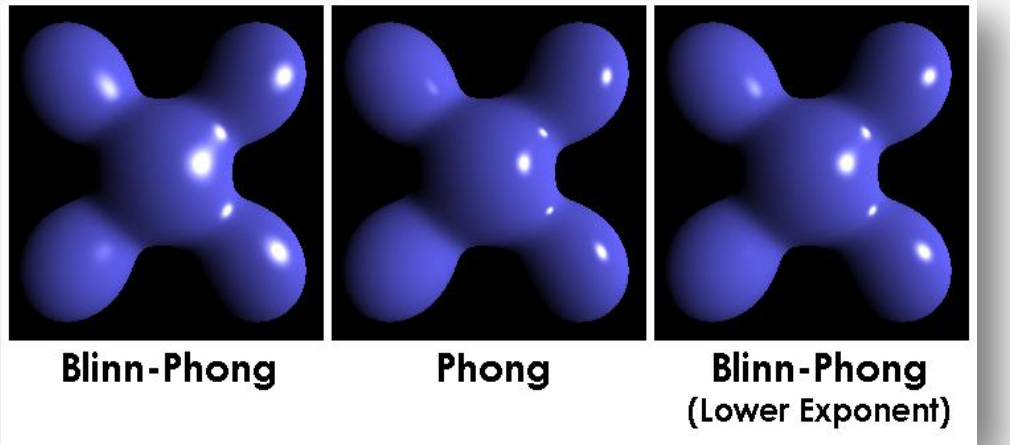
Per-vertex: normalna wierzchołka, pozycja wierzchołka

Per-light: pozycja światła, kolor (ambient, diffuse, specular)

Podstawowy model oświetlenia: Blinn-Phong



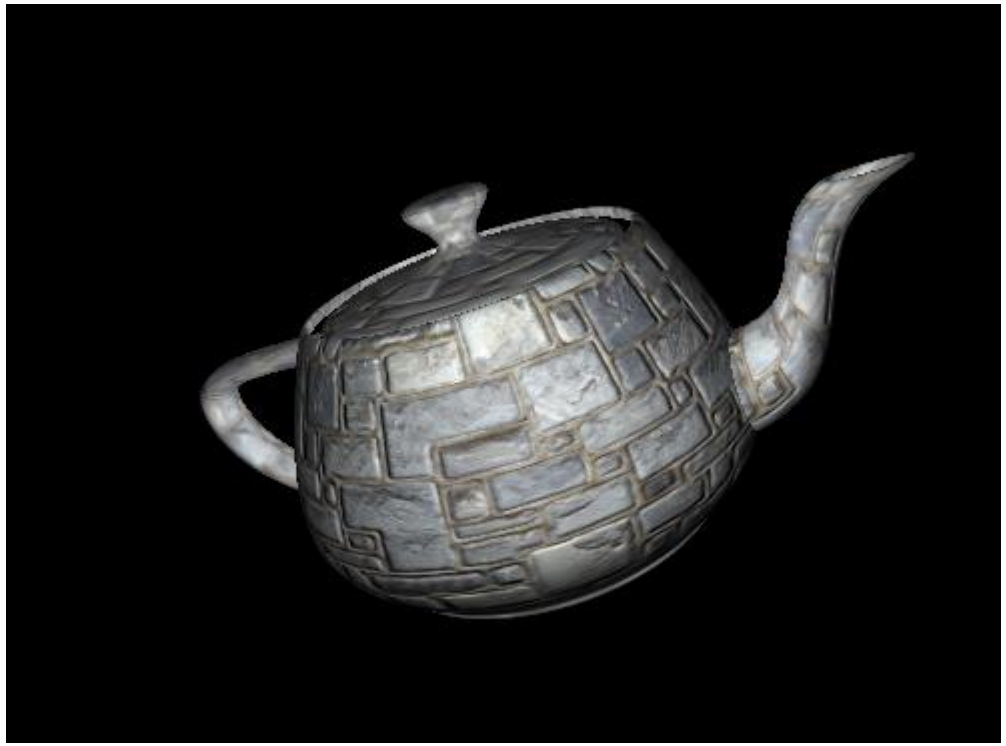
- Drobna modyfikacja pozwala nam na optymalizację shadera (Phong -> Blinn Phong)
- Operacja odbicia wektora jest bardziej kosztowna niż dodawanie wektorów
- Zamiast liczyć wektor $\vec{R}$ (odbicie) liczymy tzw. half-vector jako $\vec{H} = normalize(\vec{L} + \vec{V})$
- Zamiast liczyć $(\vec{R} * \vec{V})$ liczymy $(\vec{N} * \vec{H})$ – daje całkiem dobre przybliżenie modelu Phonga



gamedesire® academy

Toon shading

Normal mapping

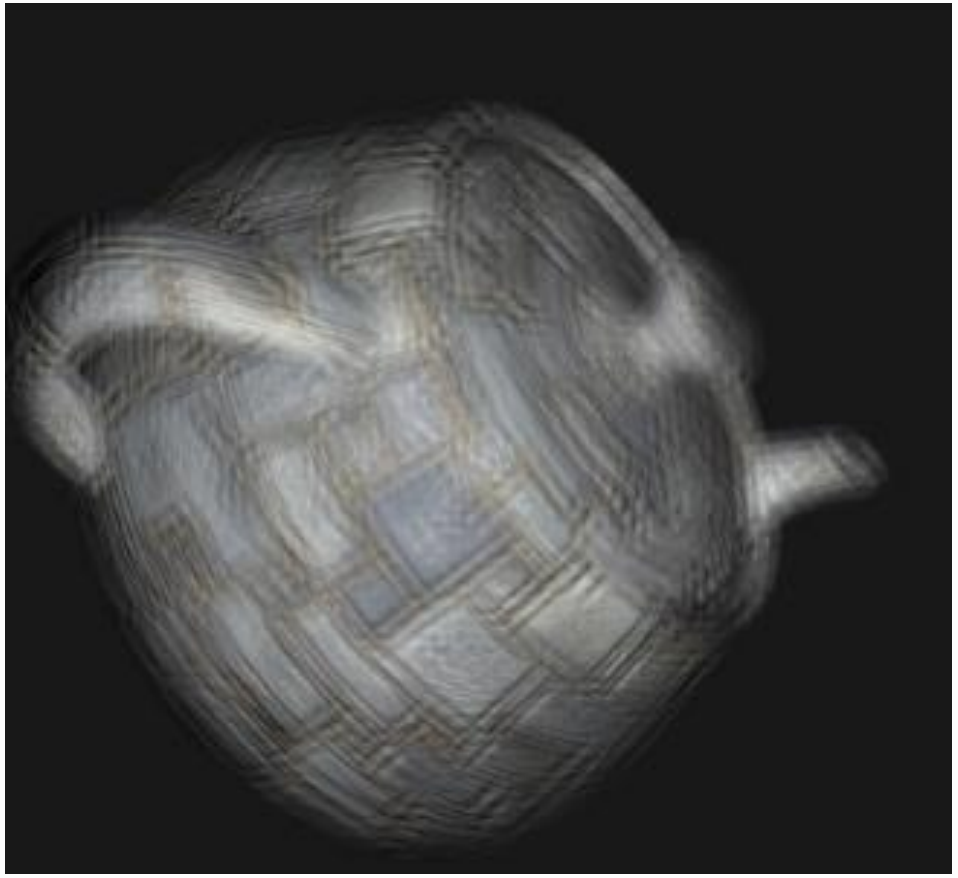


Post-processing

1. W pierwszym przebiegu renderujemy scenę do tekstury
2. Używamy tak stworzonej tekstury do teksturowania prostokąta o wymiarach całego ekranu
3. Używamy pixel shadera do stworzenia efektu post-porcessingu

gamedesire® academy

Post-processing



Unity

Standard shader (physically based shading)

- Energy conservation
- HDR
- Fresnell refraction

Surface Shaders

Vertex and Fragment shaders

Shader Lab – sposób opisu shadera w Unity

References

ftp://download.nvidia.com/developer/cuda/seminar/TDCI_Arch.pdf

https://en.wikipedia.org/wiki/High-dynamic-range_imaging

http://docs.unity3d.com/Manual/ShadersOverview.html

http://docs.unity3d.com/Manual/SL-SurfaceShaders.html

http://docs.unity3d.com/Manual/SL-ShaderPrograms.html

http://www.opengl-tutorial.org

http://developer.amd.com/tools-and-sdks/archive/legacy-cpu-gpu-tools/rendermonkey-toolsuite/

gamedesire® academy

Dziękuję

PYTANIA?