

Modelowanie materiałów

Pracownia Gier w OpenGL

Michał Drobot
Drobot.org

Cel

- Projektowanie i wdrażanie shaderów
 - Reprezentujących realistyczne materiały
 - Atrakcyjnych wizualnie
 - Wydajnych
 - NPR - non photo realistic - nierealistycznych

Plan

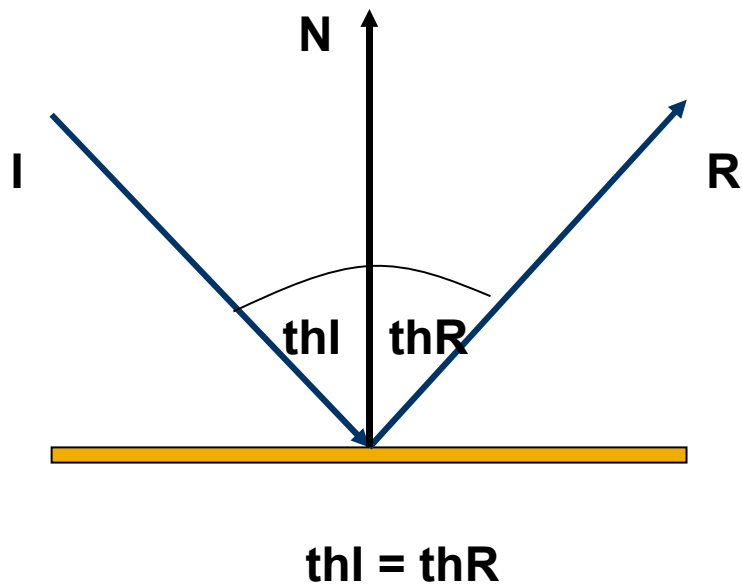
- Właściwości fizyczne materiałów (1 część)
- Właściwości interakcji materiał - światło (2 część)
- Integracja modeli (2 część)
 - Forward rendering
 - Deferred rendering

Właściwości fizyczne materiałów

- S to sujemy modele analityczno empiryczne nastawione na wydajność
- Podstawowe właściwości
 - Refleksje
 - Refrakcje
 - Współczynnik Fresnela
 - Nierówności powierzchni
 - Makroskala
 - Mikroskala

Refleksje

■ Refleksje



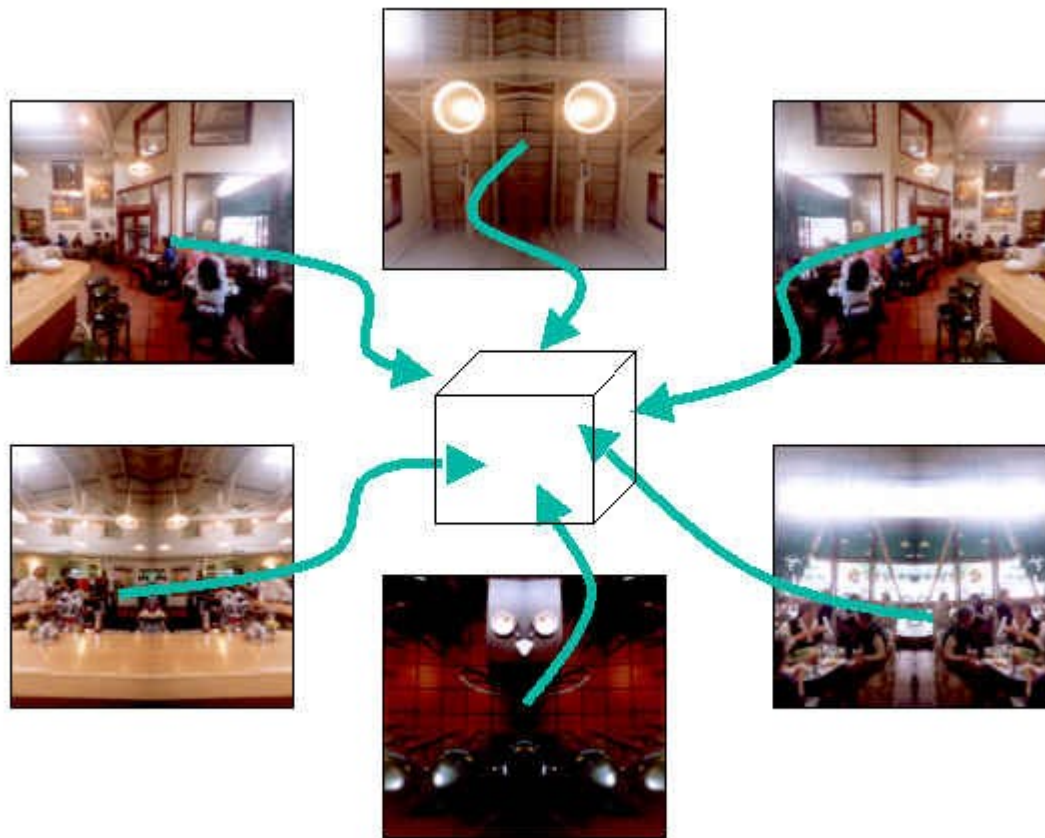
$$R = I - 2 * N * \text{dot}(I \cdot N)$$

$$R = \text{reflect}(I, N);$$

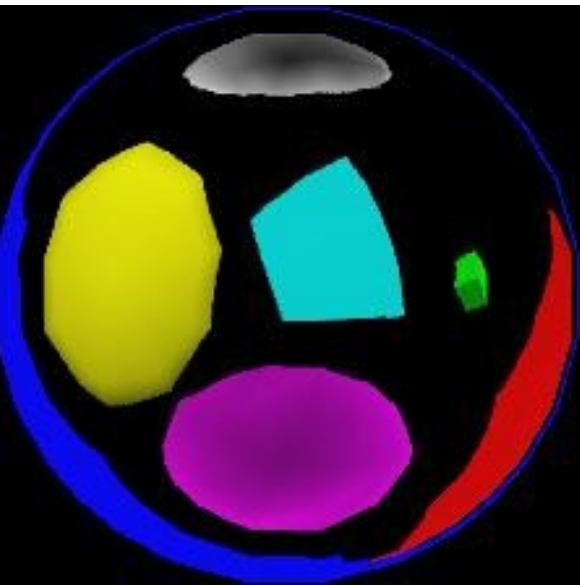
Refleksje

- Dla optymalizacji przestrzeń dookoła (otoczenie) zapisujemy i odtwarzamy przez przekształcenia
 - Sześcienne - cube mapping
 - 6 rzutów przestrzeni dookoła mapujemy na sześcian ($x, -x, y, -y, z, -z$)
 - Brak zniekształceń
 - Wsparcie sprzętowe (textureCUBE)
 - 6-krotny koszt renderingu sceny z punktu widzenia kamery
 - Paraboloidalne - (dual) paraboloid mapping
 - Mapowanie otoczenia na paraboloidy (2 'półkule')
 - Małe zniekształcenia
 - Brak wsparcia sprzętowego (ręcznie w VS/PS)
 - 2-krotny koszt renderingu sceny
 - Sferyczne
 - Mapowanie otoczenia na kulę
 - Duże zniekształcenia
 - Brak wsparcia sprzętowego
 - Pojedynczy koszt renderowania

Refleksje



Refleksje



Refleksje

```
varying vec3 E;  
varying vec3 N;  
  
void main()  
{  
    gl_TexCoord[0] = gl_MultiTexCoord0;  
  
    vec4 Vertex_ModelView = gl_ModelViewMatrix * gl_Vertex;  
    N = normalize(gl_NormalMatrix * gl_Normal);  
  
    E = vec3(Vertex_ModelView);  
    E = normalize(E);  
  
    gl_Position = ftransform();  
}
```

Refleksje

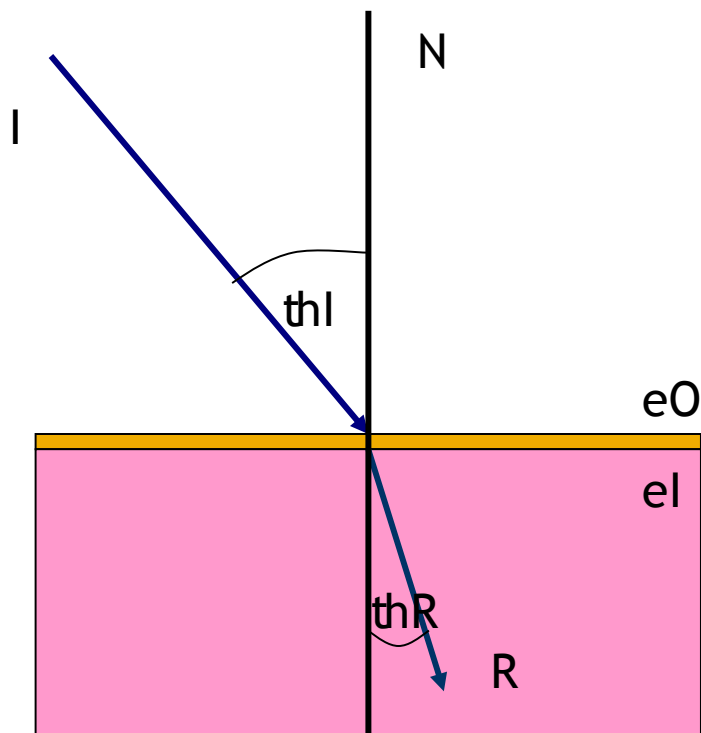
```
void main()
{
    //
    // Reflection
    //
    vec3 reflection = textureCube(enviromentCube, reflect(E,N)).rgb;

    vec3 finalColor = reflection;

    gl_FragColor = vec4(finalColor, 1.0);
}
```

Refrakcje

■ Refrakcje



Prawo Snella

$$e1 * \sin(thI) = e0 * \sin(thR)$$

$$thR = \sin^{-1}(e0 / e1 * \sin(thI))$$

Próżnia	1.0
Powietrze	1.0003
Woda	1.333
Szkło	1.5
Plastik	1.6
Diament	2.417

Refrakcje

```
vec3 ref(vec3 I, vec3 N, float e0, float e1)
{
    float eta = e0 / e1;
    float cos_theta1 = dot(-I, N);
    float cos_theta2 = 1.0 - eta * eta * ( 1.0 - (cos_theta1 * cos_theta1));
    vec3 refraction = (eta * I) + (eta * cos_theta1 - sqrt(abs(cos_theta2))) * N;

    return refraction * (vec3)(cos_theta2 > 0.0);
}
vec3 refract = refract(I, N, e0 / e1);
```

Refrakcje

```
varying vec3 E;  
varying vec3 N;  
  
void main()  
{  
    gl_TexCoord[0] = gl_MultiTexCoord0;  
  
    vec4 Vertex_ModelView = gl_ModelViewMatrix * gl_Vertex;  
    N = normalize(gl_NormalMatrix * gl_Normal);  
  
    E = vec3(Vertex_ModelView);  
    E = normalize(E);  
  
    gl_Position = ftransform();  
}
```

Refrakcje

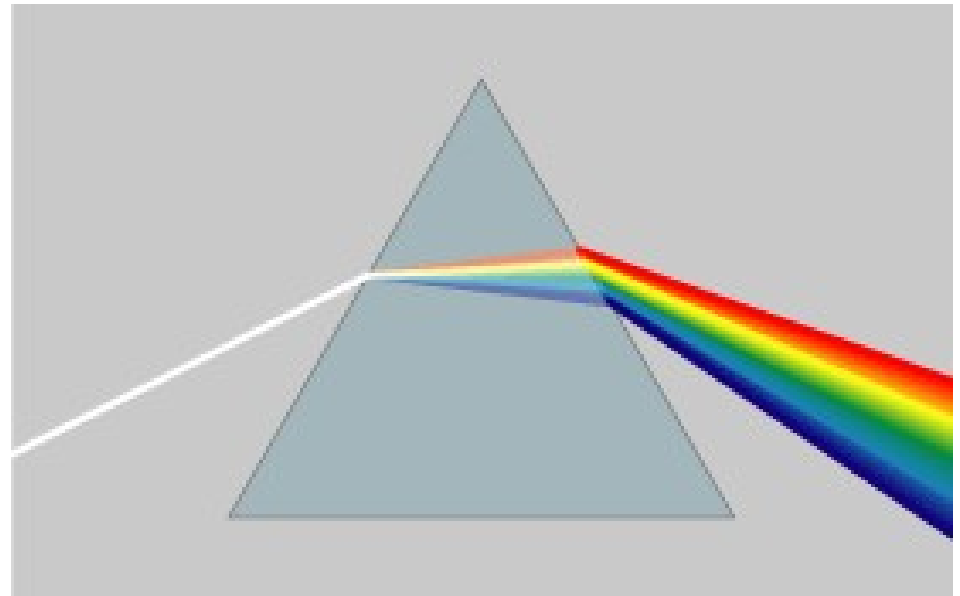
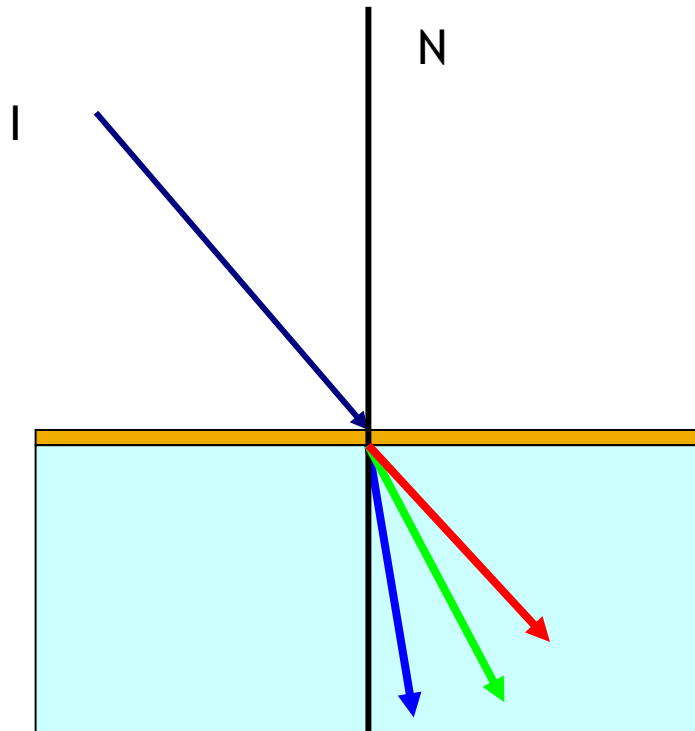
```
void main()
{
    //
    // Refraction
    //
    vec3 refraction = textureCube(enviromentCube, refract(E ,N ,eO /eI)).rgb;

    vec3 finalColor = refraction;

    gl_FragColor = vec4(finalColor, 1.0);
}
```

Refrakcje - dyfrakcja

■ Dyfrakcja

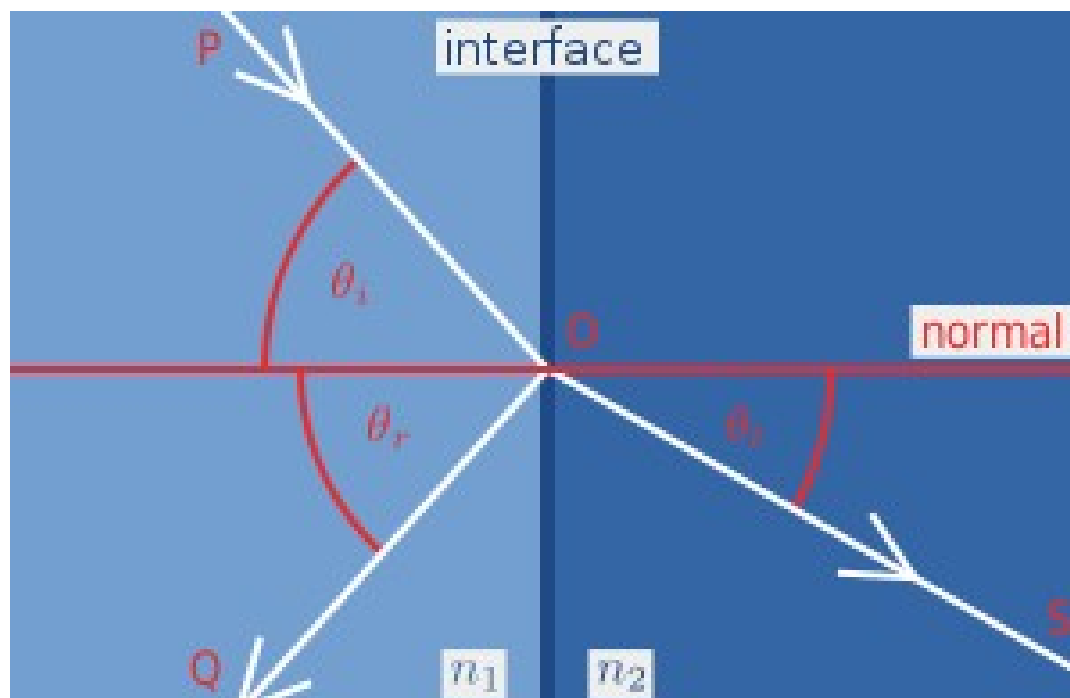


Refrakcije - abberacije

- Do domu ;]

Równanie Fresnela

■ Równanie Fresnela



$$R_s = \left[\frac{\sin(\theta_t - \theta_i)}{\sin(\theta_t + \theta_i)} \right]^2 = \left[\frac{n_1 \cos(\theta_i) - n_2 \cos(\theta_t)}{n_1 \cos(\theta_i) + n_2 \cos(\theta_t)} \right]^2 = \left[\frac{n_1 \cos(\theta_i) - n_2 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta_i\right)^2}}{n_1 \cos(\theta_i) + n_2 \sqrt{1 - \left(\frac{n_1}{n_2} \sin \theta_i\right)^2}} \right]^2$$

Równanie Fresnela

- Podczas przechodzenia przez płaszczyzną ośrodków część światła ulega odbiciu (refleksja) a część przechodzi do wnętrza ośrodka (refrakcja / światło przepuszczone)
- Woda
 - Pod niskimi kątami patrzenia dużo odbić, mało refrakcji
 - Pod dużym kątem dużo refrakcji, mało odbić

Równanie Fresnela

- Aproksymacja empiryczna

```
Float R0 = pow(1.0-refractionIndexRatio, 2.0) /  
           pow(1.0+refractionIndexRatio, 2.0);
```

```
fresnel = R0 + (1 - R0) * (1 + dot( I , N ))^pow)
```

```
C = mix ( refraction , reflection , fresnel);
```

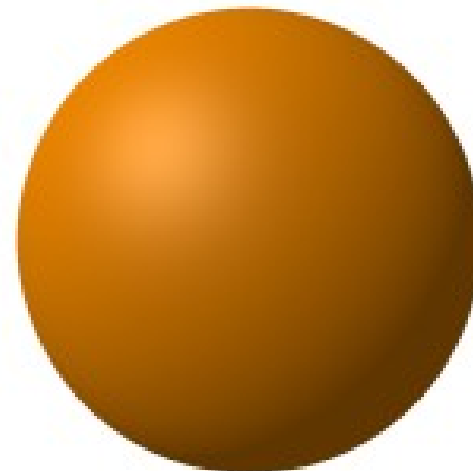
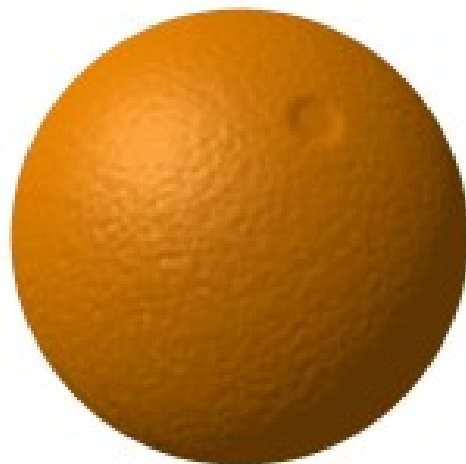
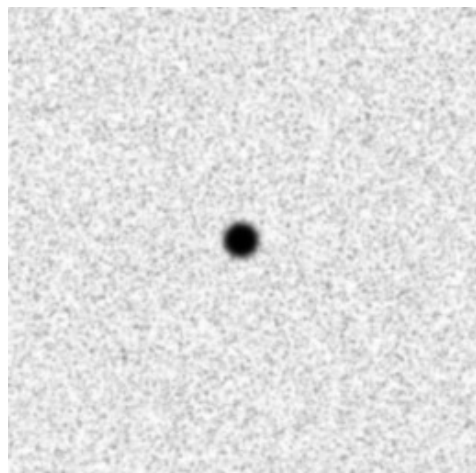
Równanie Fresnela

- Do domu ;]

Nierówności powierzchni

- Nierówności powierzchni
 - nierówności w makroskali przestrzeni renderowania wpływają na odbiór powierzchni (widzimy nierówność ‘gołym okiem’)
 - Modelujemy poprzez zniekształcenie powierzchni widzialnej oraz normalnych płaszczyzny
 - nierówności w mikroskali przestrzeni renderowania wpływają na odbiór interakcji powierzchnia - światło (nie widzimy nierówności lecz dostrzegamy jej efekt)
 - Modelujemy w trakcie analizy modelu oświetlenia

Nierówności powierzchni



Real Image

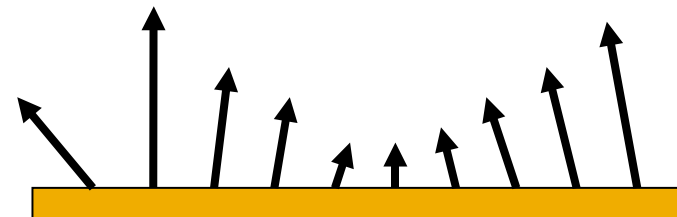
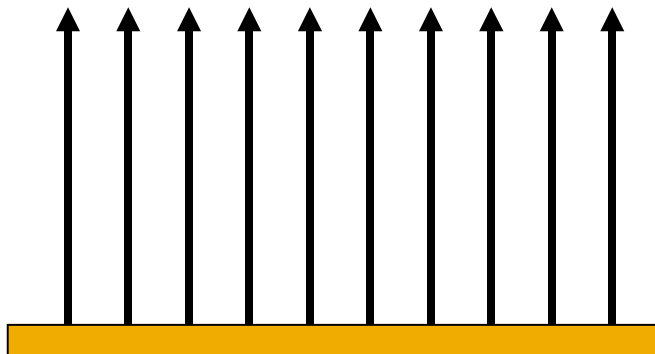
Lambertian Model

Oren-Nayar Model

Nierówności powierzchni

- Wpływ nierówności powierzchni
 - Efekt paralaksy
 - Lokalne zmiany w oświetleniu
 - Sylwetkę/obrys obiektu
 - Cień
 - Rzucany
 - Na siebie (self shadowing)
 - Na inne obiekty
 - Odbierany

Nierówności powierzchni



Bump mapping

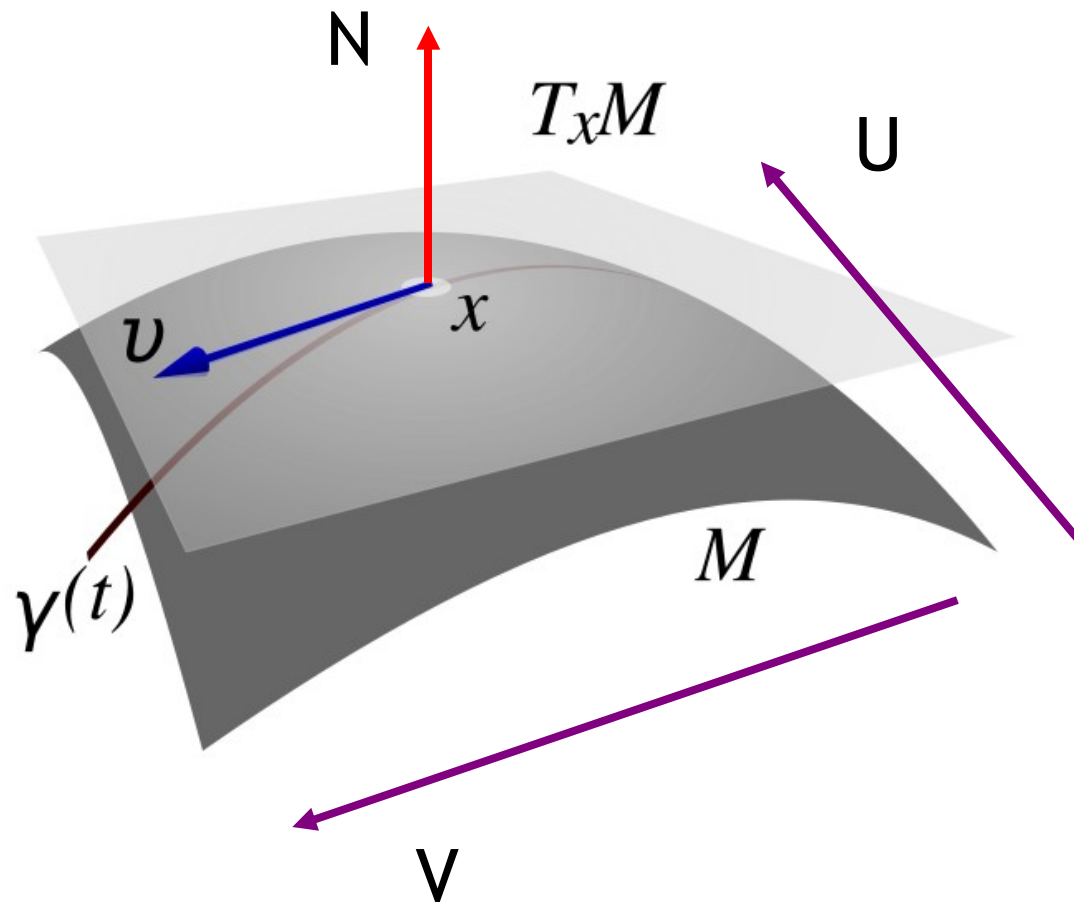
- Bump mapping
 - Podmiana istniejącej normalnej obiektu w punkcie na normalną zdefiniowaną z góry
 - Wyliczona z modelu o większej złożoności
 - Przetworzoną przez lokalne nierówności powierzchni
 - Szybki obliczeniowo
 - Wpływa jedynie na zmianę normalnych = modelowanie oświetlenia
 - Pamięciowo
 - Bump mapping = height map (8A /16A)
 - Narzut obliczeniowy
 - Normal mapping = normal map (RGB8)
 - Narzut pamięciowy
 - Podatne na artefakty przy zachowaniu min. Narzutu pamięciowego

Bump mapping

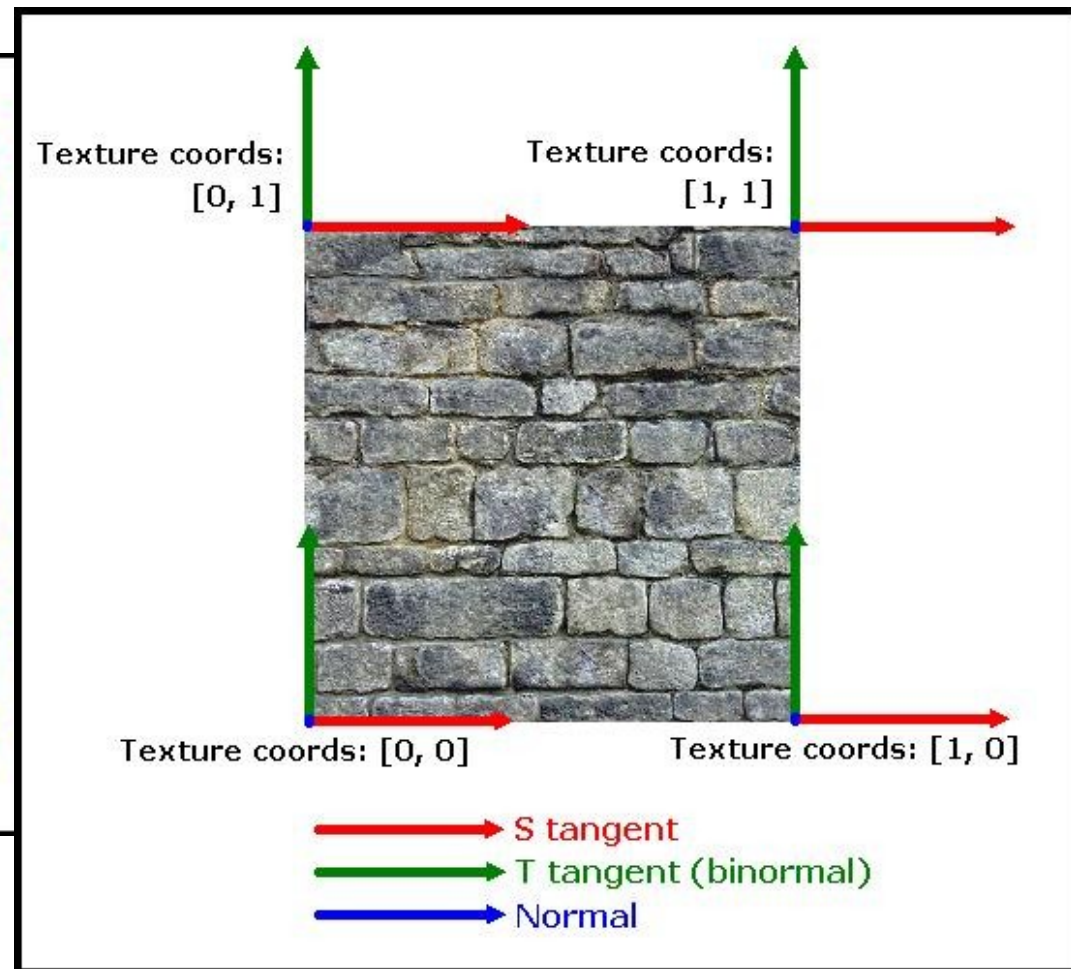
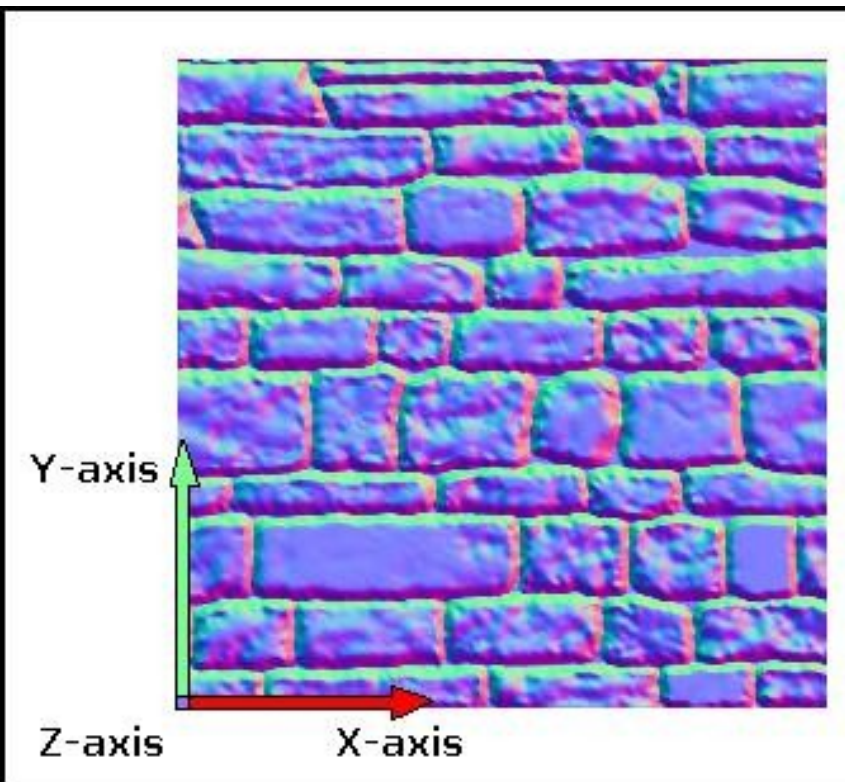
- Tangent space
 - Przestrzeń płaszczyzny obiektu
 - Wyznaczana przez
 - Wektor normalnej w punkcie
 - Wektor pochodnej koordynatów textury
 - Traktujemy jak 3D owinięcie (wrap) powierzchni obiektu
 - „płaska” powierzchnia posiada w dowolnym punkcie wektor normalny $(0,0,1)$ - do góry

Bump mapping

- Tangent space



Bump mapping

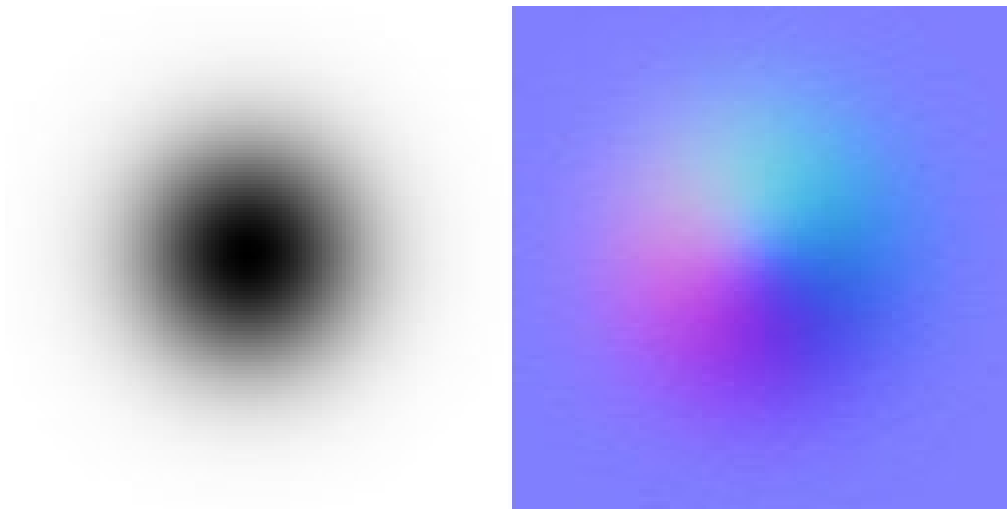
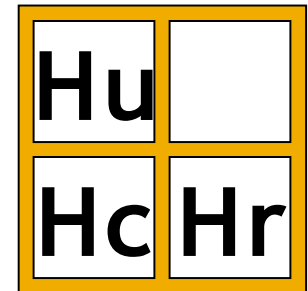


Bump mapping

- Wylizanie normalnej z height mapy

$$\text{Normal} = \frac{(H_c - H_r, H_c - H_u, 1)}{|(H_c - H_r, H_c - H_u, 1)|}$$

`Normal = normalize(vec3(ddx, ddy, 1));`



Bump mapping

- Przechowywanie normalnych
 - Normalne możemy przechowywać w teksturach
 - Większość dostępnych formatów opiera się na unsigned int
 - Wymagana ręczna kompresja z $[-1;1]$ -> $[0;1]$
 - $Compression = 0.5 * normal + 0.5;$
 - $DeCompression = 2 * (compressedNormal - 0.5)$
 - Dla zmniejszenia wymagań pamięciowych możliwość wyliczania (z) z (x,y) - koszt główny = $\sqrt{}$

Bump mapping

```
attribute vec3 tangent; attribute vec3 binormal;
varying vec3 V; varying vec3 L; varying vec3 H;

void main()
{
    gl_TexCoord[0] = gl_MultiTexCoord0;

    mat3 TBN_Matrix = gl_NormalMatrix * mat3(tangent, binormal, gl_Normal);
    vec4 Vertex_ModelView = gl_ModelViewMatrix * gl_Vertex;

    V = vec3(-Vertex_ModelView) * TBN_Matrix ;
    V = normalize(V);

    vec4 lightEye = gl_ModelViewMatrix * gl_LightSource[0].position;
    vec3 lightVec = lightEye.xyz - Vertex_ModelView.xyz;

    L = normalize(lightVec * TBN_Matrix);
    H = normalize(L + V);
    gl_Position = ftransform();
}
```

Bump mapping

```
void main()
{
    vec2 texUV = gl_TexCoord[0].xy;

    //Sample new diffuse Value

    vec3 baseColor = texture2D(baseTex, texUV).rgb;

    //Sample normal

    vec3 N = texture2D(normalTex, texUV.xy).rgb * 2.0 - 1.0;
    N = normalize(N);

    //Phong Light

    float diffuse = clamp(dot(L,N),0.0,1.0);
    float specular = pow(clamp(dot(H , N), 0.0, 1.0), 16.0);

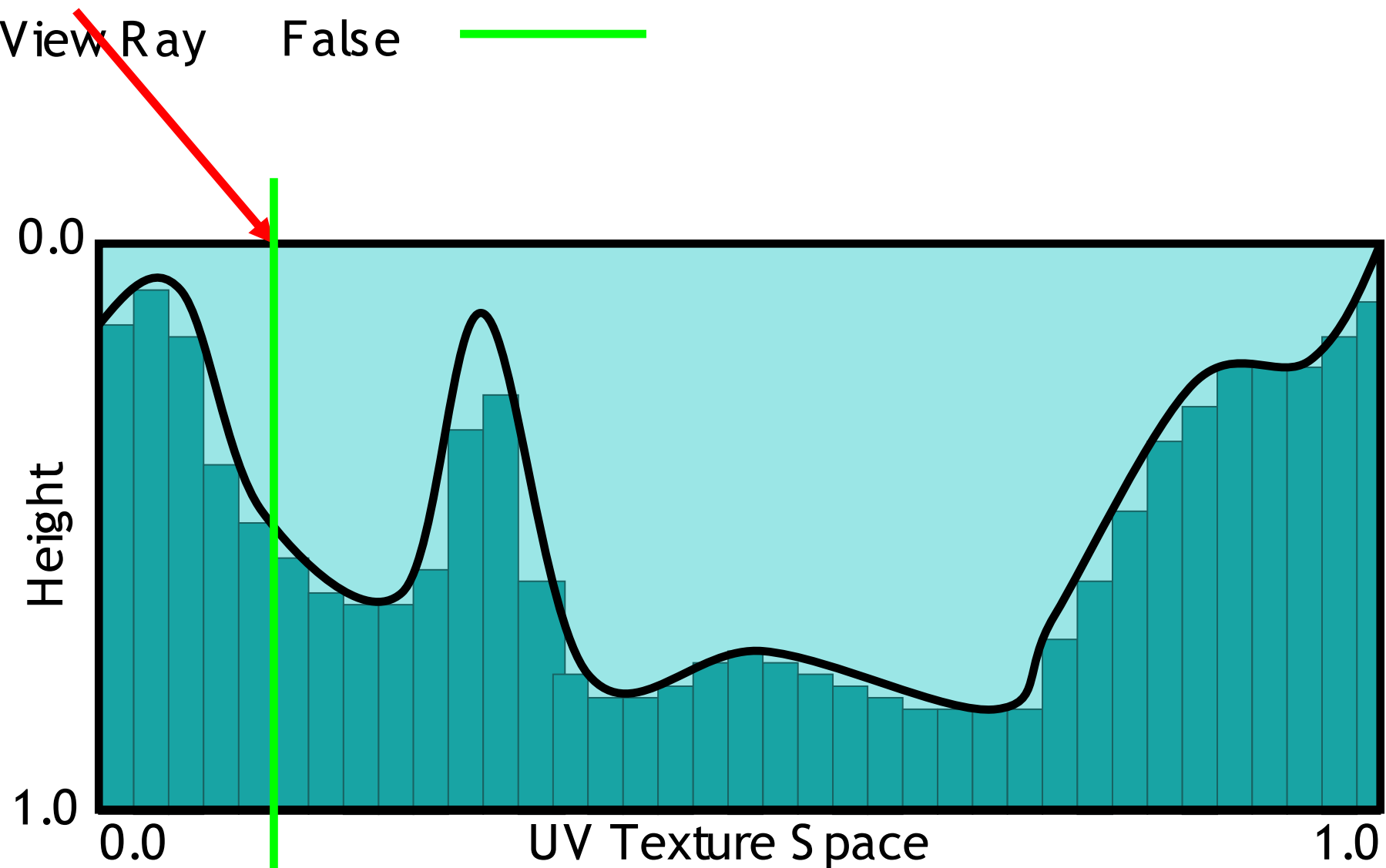
    vec3 finalColor = diffuse * baseColor.rgb + specular * baseColor.b;

    gl_FragColor = vec4(finalColor, 1.0);
}
```

Parallax mapping

View Ray

False



Parallax mapping

View Ray

False

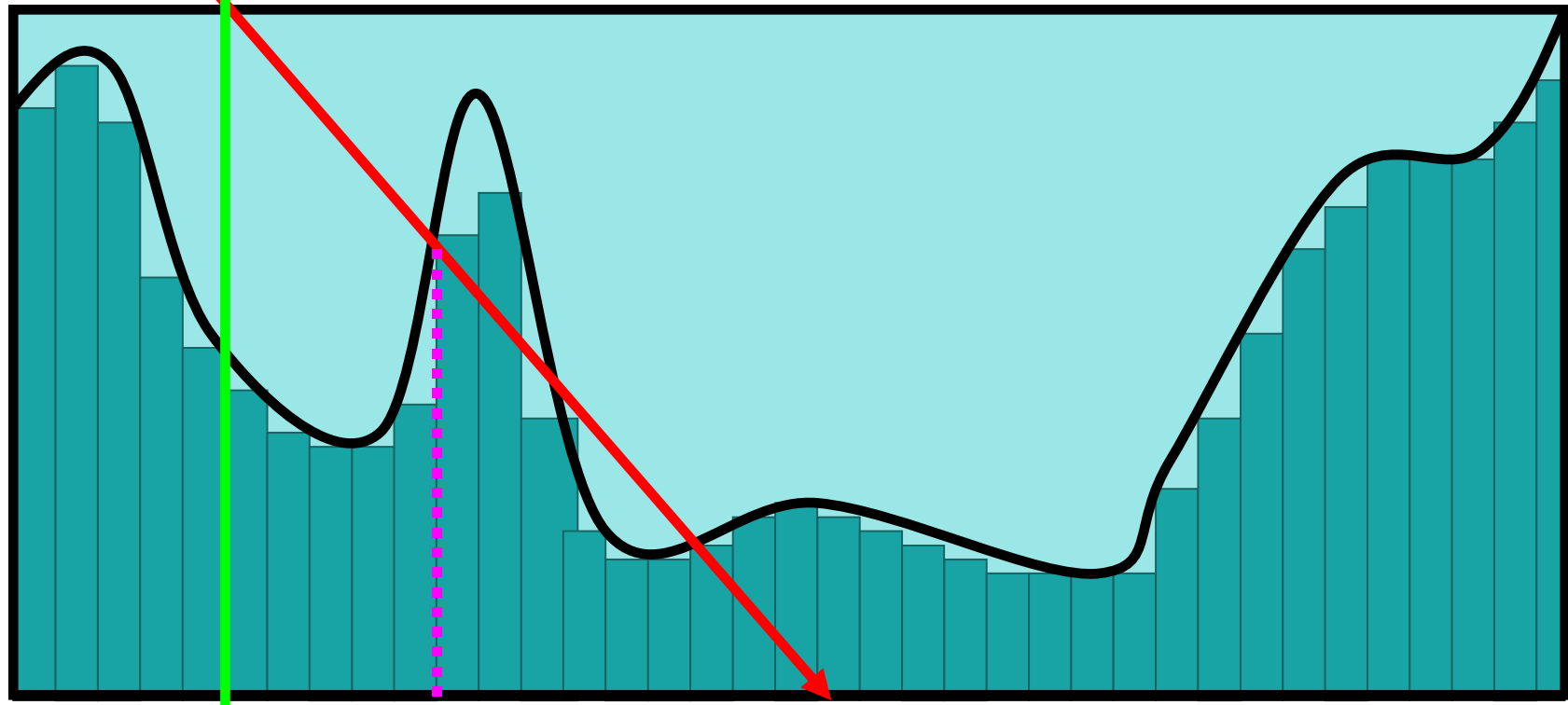
Correct



0.0

Height

1.0



0.0

UV Texture Space

1.0

Parallax mapping

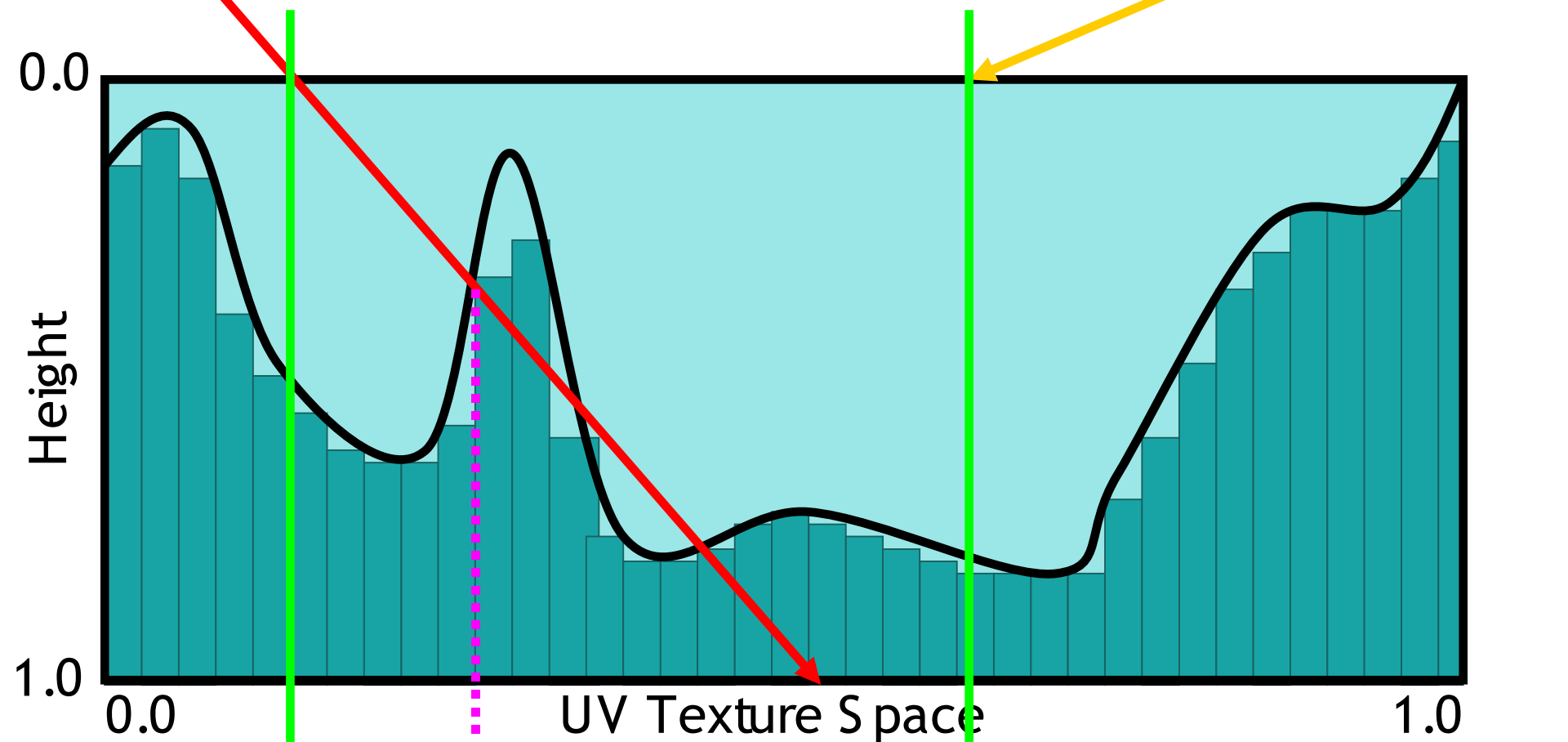
View Ray

False

Correct



Light Ray



Parallax mapping

View Ray

False



Correct

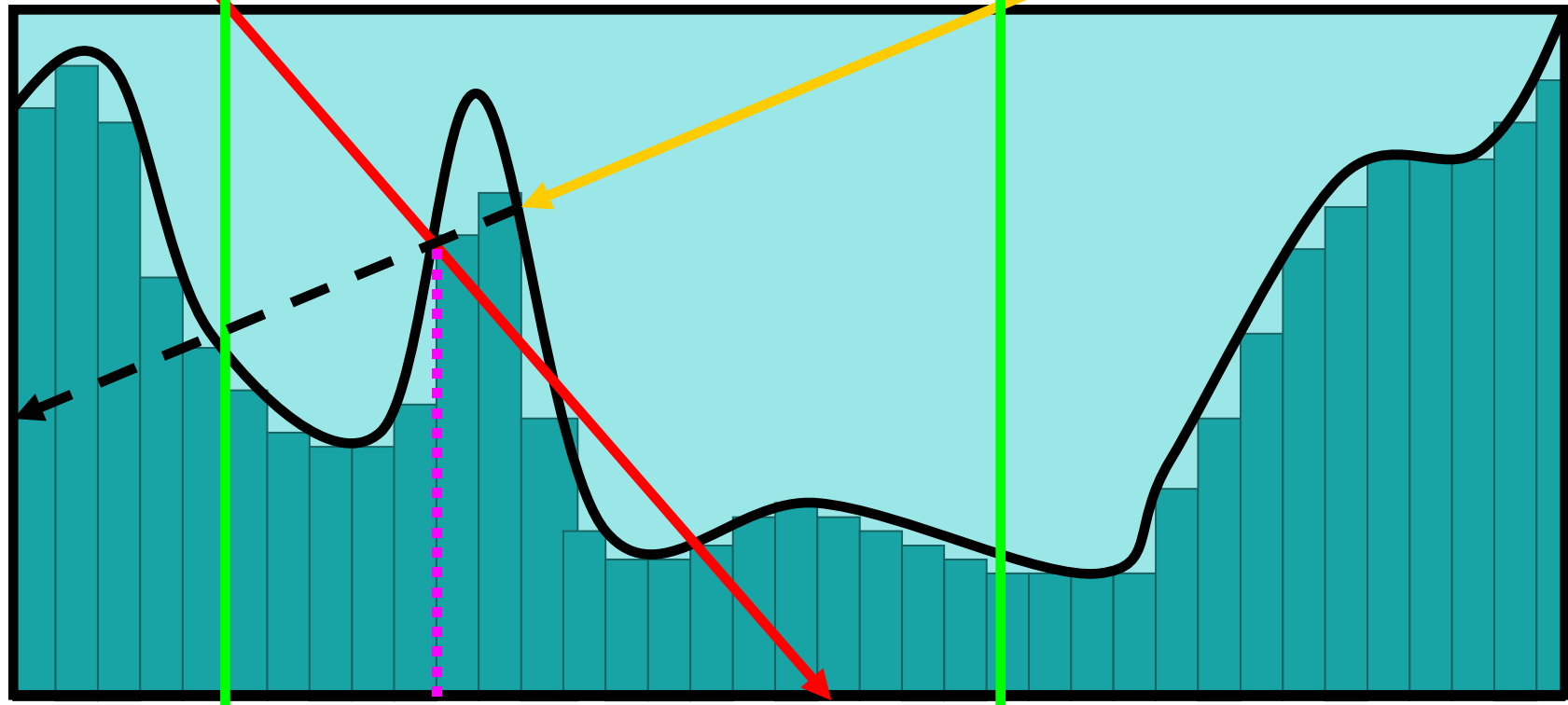


Light Ray

0.0

Height

1.0



0.0

UV Texture Space

1.0

Parallax mapping

- Parallax mapping
 - Rozszerza koncepcje bump mappingu o dodanie efektu paralaksy
 - Wymagają dodatkowej informacji o wysokości powierzchni w danym miejscu
 - Informacje o normalnych mogą być przekazana
 - Zaawansowane modele umożliwiają dodanie self-shadowingu
 - Różne metody
 - Parallax offset mapping
 - Raytraced / Raymarched Parallax
 - Relief mapping
 - Parallax Occlusion Mapping
 - Cone Step Mapping
 - Inne zaawansowane metody optymalizacyjne

Parallax mapping

- Parallax offset mapping
 - Dokonuje offsetu UV o pewną wartość zależną od wartości height mapy w danym pixelu
 - Aproksymuje prawidłową wartość offsetu
 - Powstają liczne artefakty
 - „pływanie tekstury”
 - Artefakty pod małymi kątami $\text{dot}(E, N)$
 - dystorsje
 - Mało przekonujące, proste, szybkie, w niektórych zastosowaniach warte uwagi
 - Płaskie powierzchnie

Parallax mapping

```
varying vec3 V;
```

```
void main()
```

```
{
```

```
    gl_TexCoord[0] = gl_MultiTexCoord0;
```

```
    mat3 TBN_Matrix = gl_NormalMatrix * mat3(tangent, binormal, gl_Normal);
```

```
    vec4 Vertex_ModelView = gl_ModelViewMatrix * gl_Vertex;
```

```
    //Eye vec to TS
```

```
    V = vec3(-Vertex_ModelView) * TBN_Matrix ;
```

```
    V = normalize(V);
```

```
    gl_Position = ftransform();
```

```
}
```

Parallax mapping

```
void main()
{
    vec2 texUV = gl_TexCoord[0].xy;

    //New UV calculation

    float height = texture2D(bumpTex, texUV).r;
    float v = height * scaleBias.x - scaleBias.y;
    texUV += (-V.xy * v);

    //Sample new diffuse Value

    vec3 baseColor = texture2D(baseTex, texUV).rgb;
    vec3 finalColor = baseColor;

    //Fake Occlusion biasing

    height = mix(height, 1.0, occlusionBias);
    finalColor *= height;

    gl_FragColor = vec4(finalColor, 1.0);
}
```

Parallax mapping

- Raytraced/Raymarched parallax
 - Znajduje prawidłowy offset UV poprzez raytracing
 - Na GPU w przestrzeni textury możemy jedynie aproksymować raytracing przez raymarching
 - Znajdujemy miejsce przecięcia poprzez powolne zbliżanie się do niego poprzez
 - Binary search
 - Linear Search
 - Binary + Linear Search Combo
 - Dla aktualnego pixela poszukujemy w kierunku wyznaczonym przez wektor od oka do pixela rzutowany do przestrzeni textury = Tangent Space
 - Metoda ta rozwiązuje problem
 - Paralaksy
 - Stabilności (częściowo przy zachowaniu wysokiej wydajności)
 - Metoda umożliwia rozwiązanie
 - Cieniowania
 - Sylwetek

Parallax mapping

- Relief mapping
 - Poszukuje miejsca przecięcia
 - Liniowo przy stałej liczbie kroków
 - Następnie binarnie o stałej liczbie kroków „dookoła” najlepszego dotąd miejsca
 - W zależności od ilości kroków może dość do pomyłek == artefaktów
 - W sytuacji małego kąta $\text{dot}(E, N)$ dokładność znacząco spada przy zbyt małej liczbie kroków

Parallax mapping

- Parallax Occlusion Mapping
 - Wykorzystuje SM 3.0
 - Zawiera dynamiczne ograniczenia na ilość kroków
 - Dla dużych kątów zmniejszamy, dla małych zwiększamy
 - Możliwe wiele optymalizacji

Parallax mapping

- Metody opierające się na preprocessingu danych
 - Kodują optymalną długość kroku przy użyciu wyliczonych uprzednio struktur
 - Cone Step Mapping
 - Distance Sphere Mapping
 - Etc.
 - Wydajniejsze jeśli celujemy w wysoką jakość
 - Nie do zastosowania dla height map dynamicznych z racji wymagań preprocessingu

Parallax mapping

■ Rozwiązanie problemu

■ Cieniowania

- Dokonujemy dodatkowego raytracingu wektora od światła do pixela
- Zapisujemy uzyskaną podczas raytracingu głębokość powierzchni
- Porównujemy z głębokością aktualnego pixela
- W zależności od wyniku cieniuujemy bądź nie

Parallax mapping

■ Rozwiązanie problemu

■ Sylwetki

■ SM4.0

- GS Na krawędziach obiektów tworzy dodatkowe poligony
- Vertex shader wyciąga poligony wg normalnej sylwetki
- Dla dodatkowych poligonów przeprowadzamy raytracing w przestrzeni poligonu
- W razie trafienia zapisujemy wynik jak zwykle
- W przeciwnym razie wycinamy dany pixel (tworzymy maske alpha wystającego fragmentu)

Właściwości interakcji

■ Modele oświetlenia

■ Realistyczne

- analitycznie bądź empirycznie modelują interakcje materiału o określonych właściwościach ze światłem
- Korzystamy z aproksymacji istniejących modeli matematycznych
- Wykorzystujemy dane pomiarowe materiałów rzeczywistych

■ Nierealistyczne

- Mają za zadanie przedstawić dany obiekt w sposób odpowiadający konkretnemu celowi
 - CAD
 - Blueprint
 - Medical analysis
 - Gry (toon, ink & paint..)

Właściwości interakcji

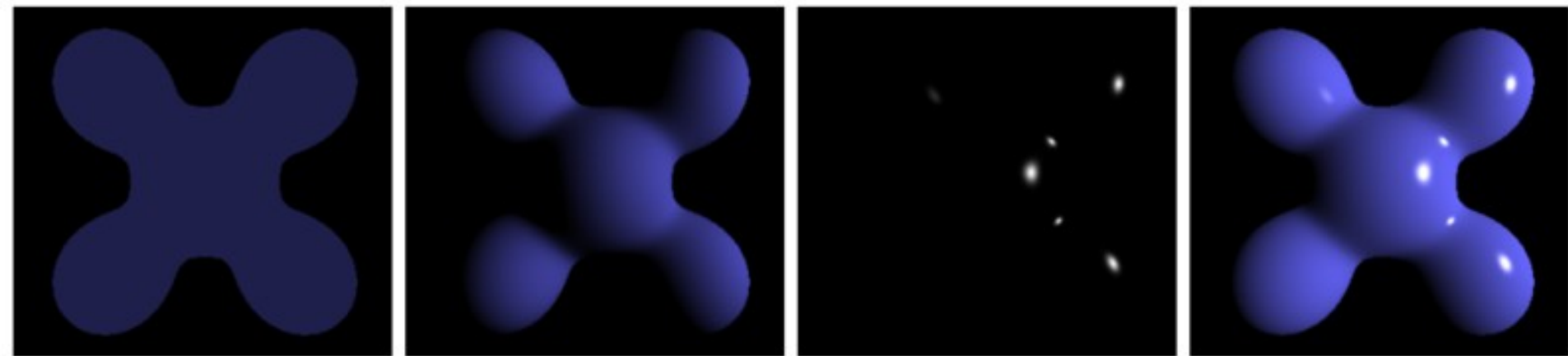
■ Realistyczne

- Właściwości materiału wpływające na interakcje
 - Mikrostruktura
 - Izotropia
 - Metal...
 - Anizotropia
 - Metal szczotkowany, włosy, satyna..
 - Przepuszczalność świetlna
 - Wosk, skóra, plastik...
 - Wielowarstwowość
 - Skóra, lakier samochodowy...

Właściwości interakcji - Phong

- Podstawowy model oświetlenia
 - Phong Shading
 - Rozdziela oświetlenie na komponenty
 - Diffuse
 - Ambient
 - Specular
 - Właściwości modelu powierzchni
 - Izotropia
 - Idealna gładkość powierzchni
 - Idealne odbicie (lustrzane)

Właściwości interakcji - Phong



Ambient

+

Diffuse

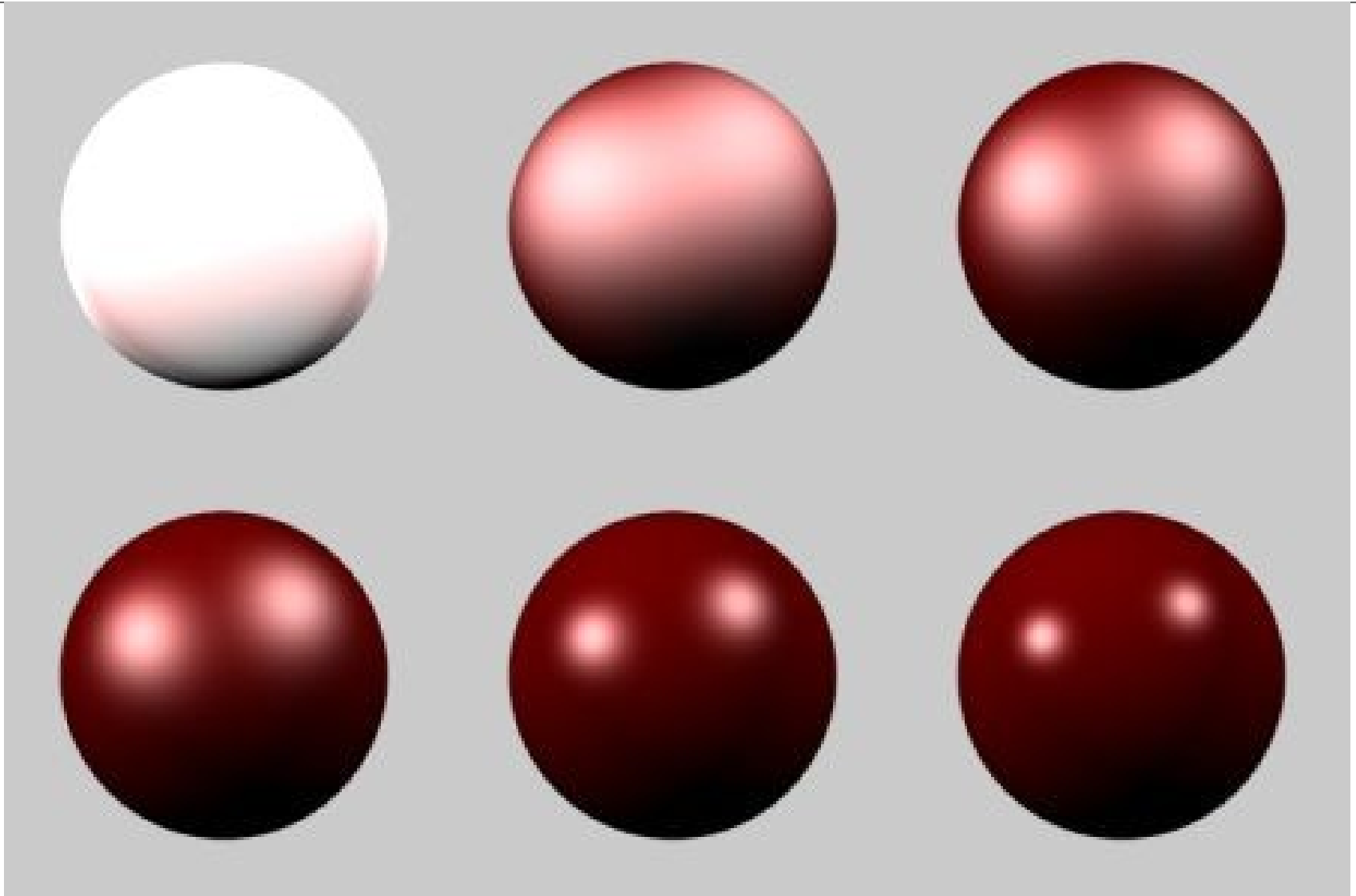
+

Specular

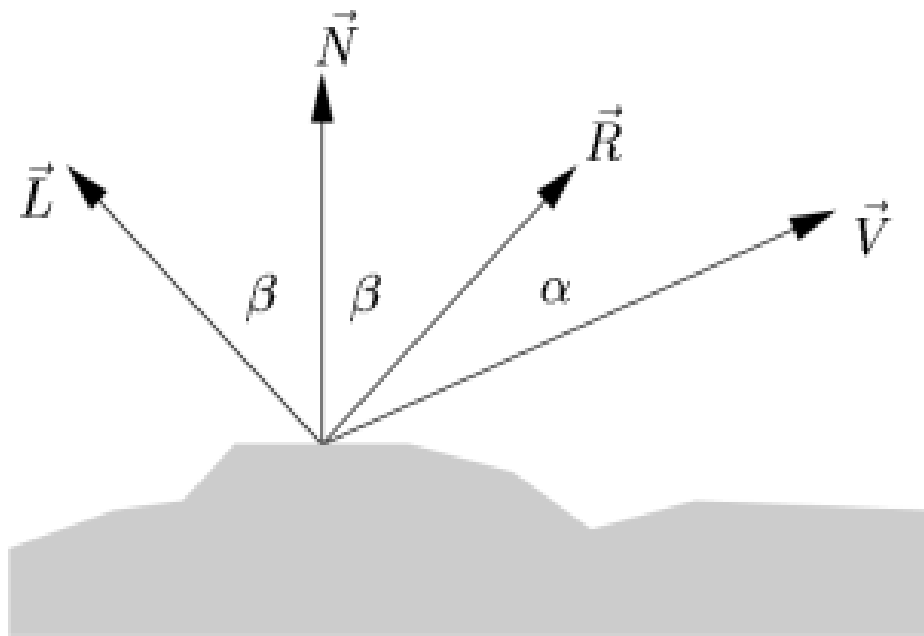
=

Phong Reflection

Właściwości interakcji - Phong



Właściwości interakcji - Phong



Color = ambient
+
diffuse
+

$$I_d = I_i \cdot \cos \beta,$$

$$I_s = I_r \cdot \cos^n \alpha,$$

$$I = k_a I_a + I_i \left(k_d (\vec{N} \cdot \vec{L}) + k_s f(\beta) (\vec{R} \cdot \vec{V})^n \right)$$

Właściwości interakcji - Phong

- Phong - Blinn
 - Zastępuje wektor R - odbicia wektorem połówkowym $H = (V + L)$
 - Przyśpiesza obliczenia
 - Wizualnie podobny rezultat

Właściwości interakcji - Phong

$\text{finalColor} = \text{emmissive} + \text{ambient} + \text{diffuse} + \text{specular}$

$\text{emmissive} = K_e$

$\text{ambient} = K_a * \text{globalAmbient}$

$\text{diffuse} = K_d * \text{lightColor} * \max(\text{dot}(\mathbf{N}, \mathbf{L}), 0)$

$\text{specular} = K_s * \text{lightColor} * \text{facing} * \text{pow}(\max(\text{dot}(\mathbf{N}, \mathbf{H}), 0), \text{shinniness})$

K_e - kolor emisji materiału

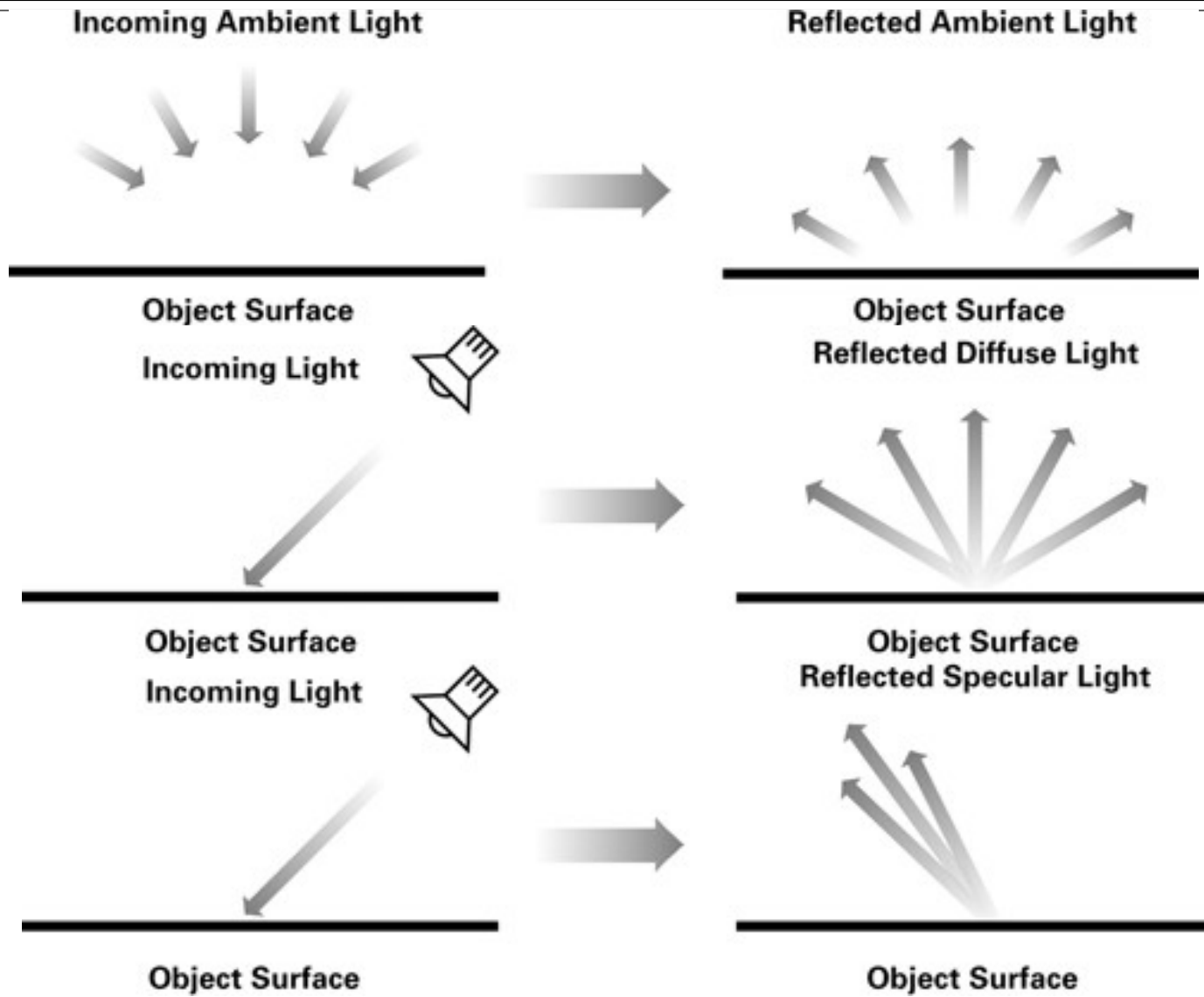
K_a - kolor odbicia światła rozproszonego

K_d - kolor właściwy powierzchni

K_s - kolor właściwy odbic

globalAmbient - globalny kolor światła rozproszonego

Właściwości interakcji - Phong



Właściwości interakcji - BRDF

■ BRDF

- Dwukierunkowa funkcja rozkładu odbicia
- Wyraża stosunek luminancji energetycznej mierzonej w kierunku obserwatora do natężenia napromienienia badanego z kierunku padania promieniowania
- w pełni charakteryzuje własności refleksyjne powierzchni odbijającej
- Można określić empirycznie poprzez pomiary interakcji światła z powierzchnią
- Dotychczasowe proste modele matematyczne : Lambert, Blinn, Phong , Phong - Blinn są uproszczonymi BRDF

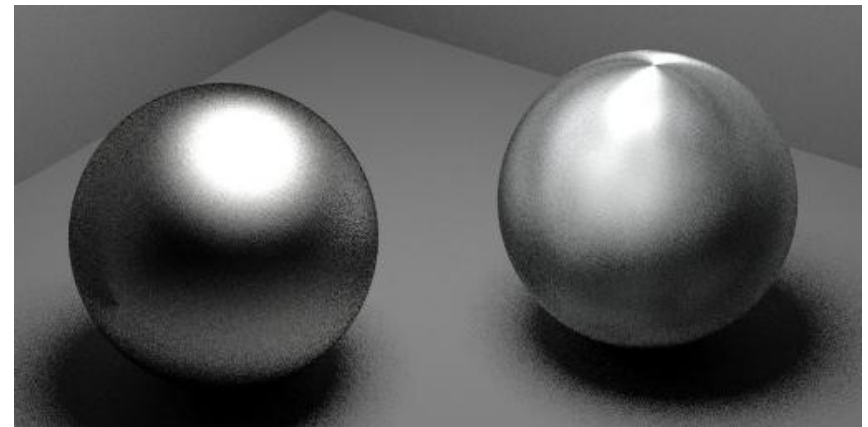
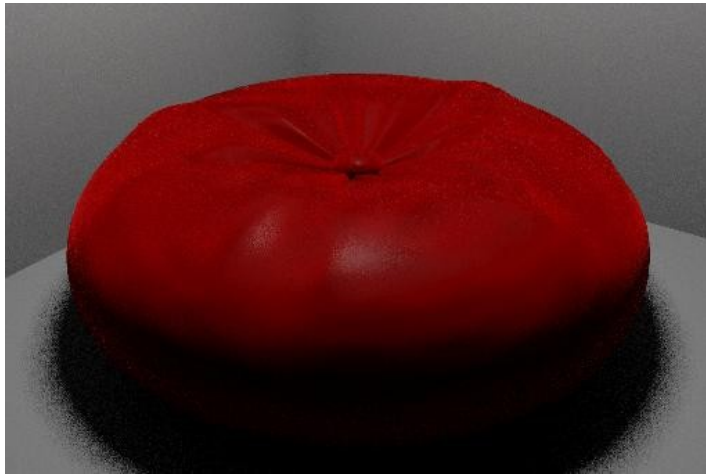
Właściwości interakcji - BRDF

■ BRDF

- W_i - wektor w kierunku światła
- W_o - wektor w kierunku kamery
- θ_i - kąt między W_i a N - normalną płaszczyzny w punkcie
- L - radiancja - energia oddawana przez materiał
- E - irradiancja - energia dochodząca do materiału

$$f_r(\omega_i, \omega_o) = \frac{dL_r(\omega_o)}{dE_i(\omega_i)} = \frac{dL_r(\omega_o)}{L_i(\omega_i) \cos(\theta_i) d\omega_i}$$

Właściwości interakcji - BRDF



Właściwości interakcji - BRDF

- Podstawowe modele BRDF - Diffuse
 - Lambert
 - $\text{Dot}(N,L)$
 - Uproszczony
 - Powierzchnia gładka
 - Oren - Nayar
 - Zakłada izotropową powierzchnię o współczynniku nierówności - szorstkości
 - Świetnie modeluje powierzchnie o zmiennych nierównościach
 - Piasek , beton , glinę...

Właściwości interakcji - Oren Nayar



Real Image

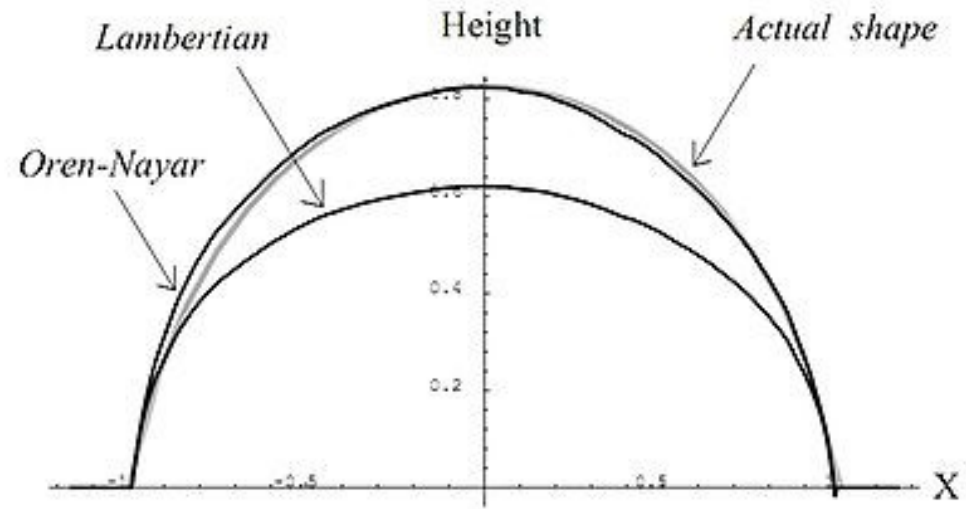
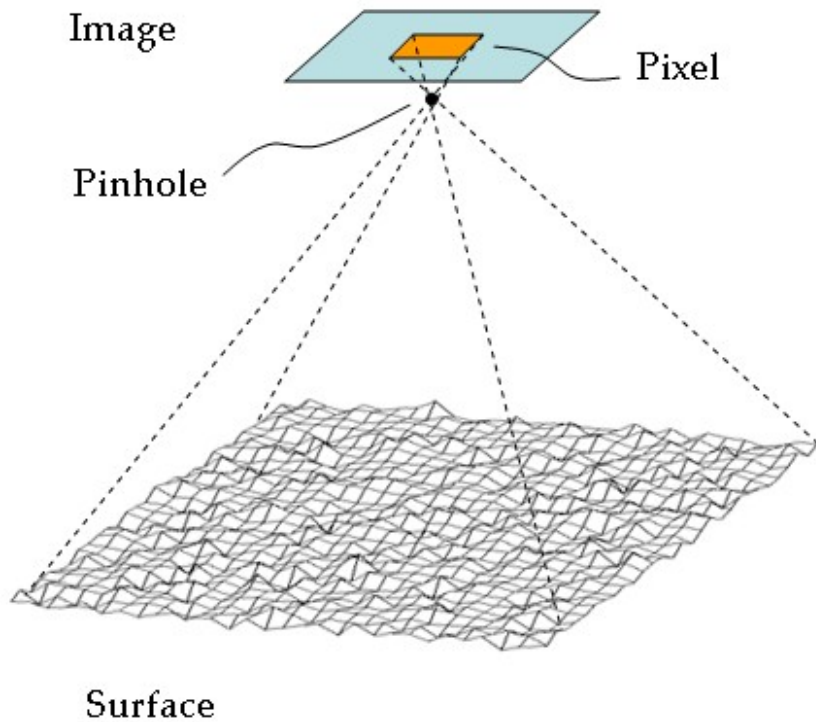


Lambertian Model



Oren-Nayar Model

Właściwości interakcji - Oren Nayar



Właściwości interakcji - Oren Nayar

$$L_r = \frac{\rho}{\pi} \cdot \cos \theta_i \cdot (A + B \cdot \max(0, \cos(\phi_i - \phi_r)) \cdot \sin \alpha \cdot \tan \beta) \cdot L_i$$

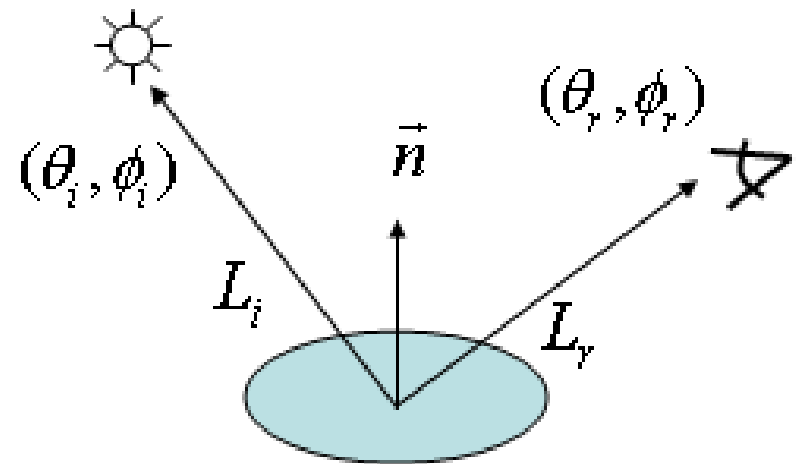
$$A = 1 - 0.5 \frac{\sigma^2}{\sigma^2 + 0.33}$$

$$B = 0.45 \frac{\sigma^2}{\sigma^2 + 0.09}$$

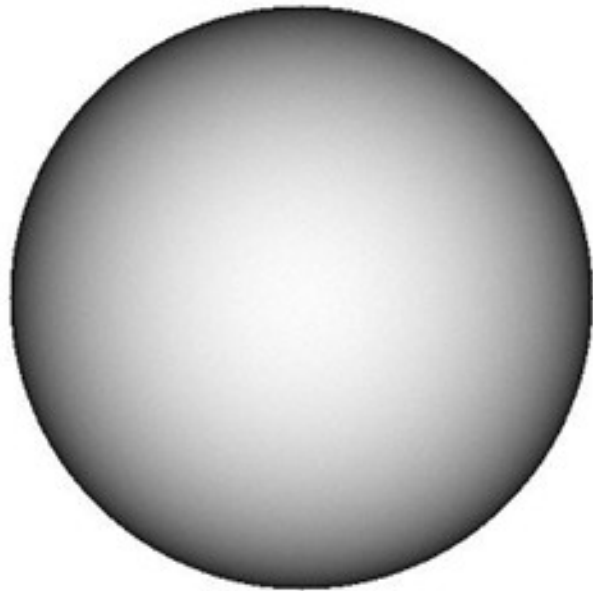
$$\alpha = \max(\theta_i, \theta_r)$$

$$\beta = \min(\theta_i, \theta_r)$$

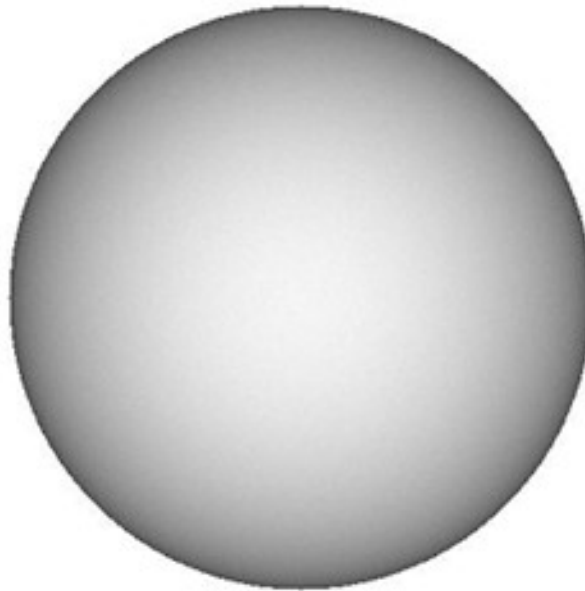
Sigma - współczynnik
nierówności



Właściwości interakcji - Oren Nayar



$$\sigma = 0$$



$$\sigma = 0.1$$



$$\sigma = 0.3$$

BRDF Specular

- Podstawowe modele BRDF - Specular
 - Phong
 - Gaussian
 - Beckmann
 - Heidrich-Seidel
 - Ward anisotropic
 - Cook-Torrance

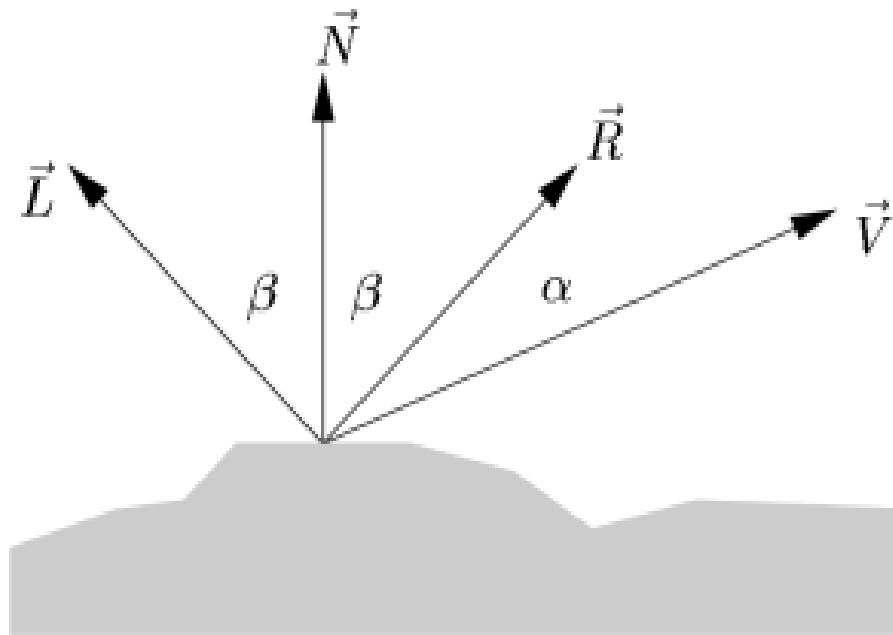
BRDF Specular

■ Phong

- Zakłada lustrzane odbicie
- Izotropową powierzchnię
- Blinn-Phong - zamienia R na H
- N - kontroluje gładkość

$$k_{\text{spec}} = (\hat{R} \cdot \hat{V})^n$$

$$k_{\text{spec}} = (\hat{N} \cdot \hat{H})^n$$



BRDF Specular

- Gaussian
 - Lepsze rezultaty niż Phong
 - Wolniejsze
 - Założenia jak w Phongu
 - M - kontroluje gładkość [0;1]

$$k_{\text{spec}} = e^{-\left(\frac{\angle(N, H)}{m}\right)^2}$$

BRDF Specular

- Beckmann
 - Model fizyczny
 - Bardzo dokładny
 - Kosztowny
 - m kontroluje gładkość

$$k_{spec} = \frac{1}{4m^2 \cos^4(N, H)} e^{-\left(\frac{\tan(N, H)}{m}\right)^2}$$

BRDF Specular

■ Heidrich-Seidel

- Model przewiduje anizotropową powierzchnię
 - Włosy, szczotkowany metal...
- N - wykładnik anizo
- T - wektor równoległy do nierówności powierzchni
 - Dla włosów będzie to kierunek ułożenia pojedynczego włosa
- T - możemy zapisywać na teksturze w Tangent Space podobnie jak wektory normalnych

$$k_{spec} = [\sin(L, T) \sin(V, T) - \cos(L, T) \cos(V, T)]^n$$

BRDF Specular



BRDF Specular

- Ward anisotropic
 - Model parametryczny
 - Dokładny
 - Kosztowny
 - Parametry α_x , α_y kierują anizo wg zadanych w płaszczyźnie normalnej wektorów ortogonalnych X i Y (analogicznie do T z Heidricha-Seidela)

$$k_{spec} = \frac{1}{\sqrt{(N \cdot L)(N \cdot V)}} \frac{N \cdot L}{4\alpha_x\alpha_y} \exp \left[-2 \frac{\left(\frac{H \cdot X}{\alpha_x}\right)^2 + \left(\frac{H \cdot Y}{\alpha_y}\right)^2}{1 + (H \cdot N)} \right]$$

BRDF Specular

■ Cook-Torrance

- Najbardziej zaawansowany model analityczny
- Wg tego modelu dokonuje się pomiarów większości pomiarów BRDF powierzchni
- Zadana przez dystrybucje Beckmanna, współczynnik Fresnela oraz zanik geometryczny

$$\frac{DFG}{E \cdot N} \quad D = \frac{\exp\left(-\left(\frac{\tan \alpha}{m}\right)^2\right)}{4m^2 \cos^4 \alpha} \quad \text{Alfa - kąt między H i N}$$
$$F = (1 + E \cdot N)^\lambda$$
$$G = \min\left(1, \frac{2(H \cdot N)(E \cdot N)}{E \cdot H}, \frac{2(H \cdot N)(L \cdot N)}{E \cdot H}\right)$$

BRDF

- Funkcje BRDF tworzymy poprzez kombinacje odpowiednich dystrybucji rozproszenia i odbicia
- O doborze decyduje typ modelowanej powierzchni oraz wydajność łączonego rozwiązania
- Można stosować odczyt BRDF z funkcji wedle zadanych wektorów, które przyjmuje funkcja

NPR

- Non Photorealistic Rendering
 - Zastosowania
 - CAD
 - Gooch shader
 - Blueprint shader - depth peeling
 - Toon
 - Toon shader
 - Ink & Paint
 - Crosshatch

NPR - CAD

- W aplikacjach typu CAD zachodzi potrzeba oddania cech przestrzennych obiektu
 - Typowym inżynierską formą przekazu są blueprinty
 - Wydruki niebiesko - białe przecinających się krawędzi
 - Cieniowanie umożliwiające łatwiejsze dostrzeganie detali geometrycznych
 - Gooch shading

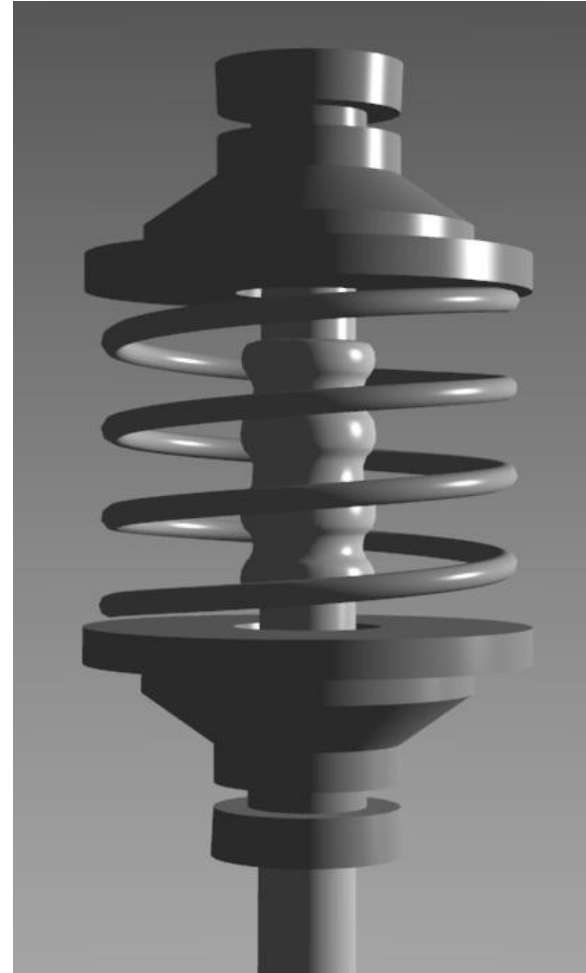
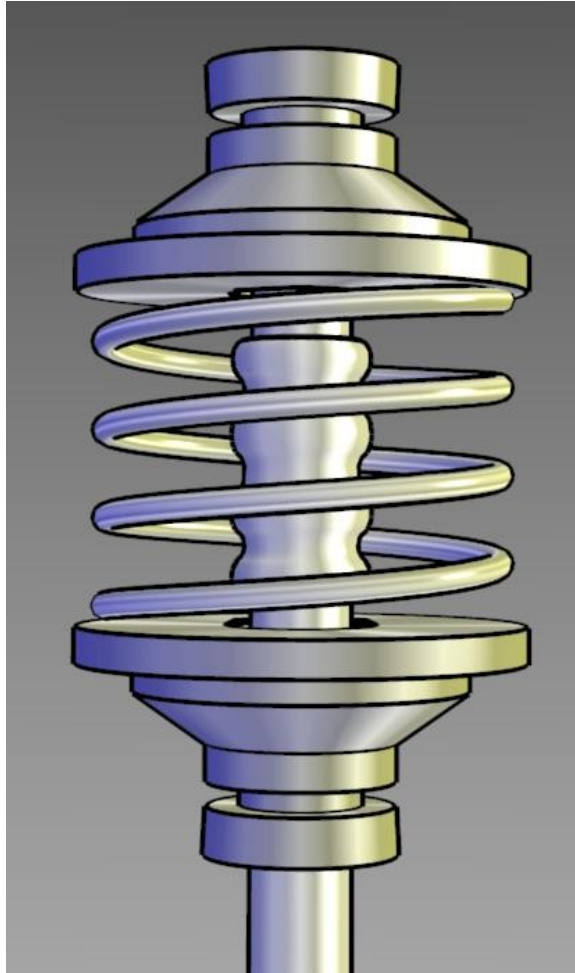
NPR - CAD - Gooch

■ Gooch shading

- Opiera się o techniki ilustracji tradycyjnej
- Wykorzystuje pełen zakres oświetlenia dodając zmieszane kolory ciepłe oraz zimne wg kątu padania światła
- Pozwala zaznaczać krawędzie sylwetek

$$I = \left(\frac{1 + \hat{i} \cdot \hat{n}}{2}\right)k_{cool} + \left(1 - \frac{1 + \hat{i} \cdot \hat{n}}{2}\right)k_{warm}$$

NPR - CAD - Gooch

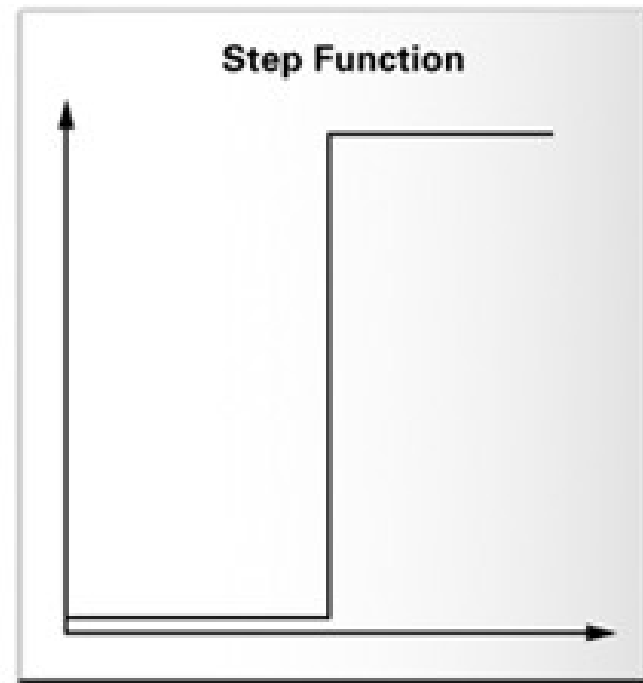
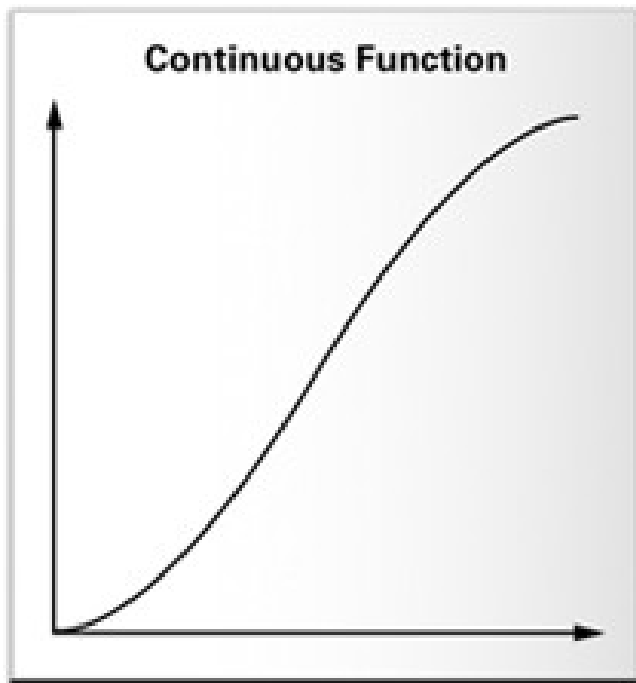


NPR - CAD - Toon

- Toon shading
 - Opisuje światło kilkoma tonami zamiast pełną paletą
 - Wymaga 3 podstawowych elementów
 - Min 2 - tonowego oświetlenia : jasno/ciemno
 - Min 1 - tonu na odbłyски jeśli są wystarczająco mocne
 - Zaznaczenia krawędzi

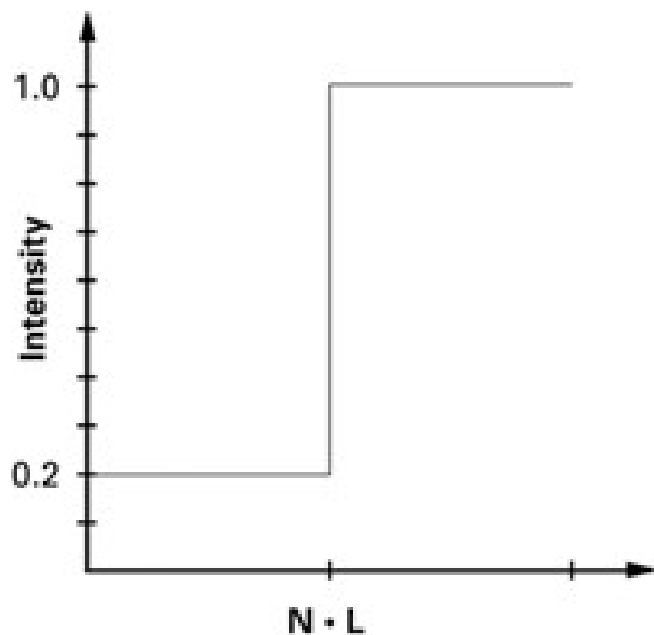
NPR - CAD - Toon

- Funkcje $\text{dot}(N, L)$ reprezentujemy schodkową funkcją przejścia



NPR - CAD - Toon

- Funkcja może być zapisana w teksturze jeśli zależy nam na większej ilości odcieni



1D Texture

`I = texture1D(stepTex, dot(N,L));`

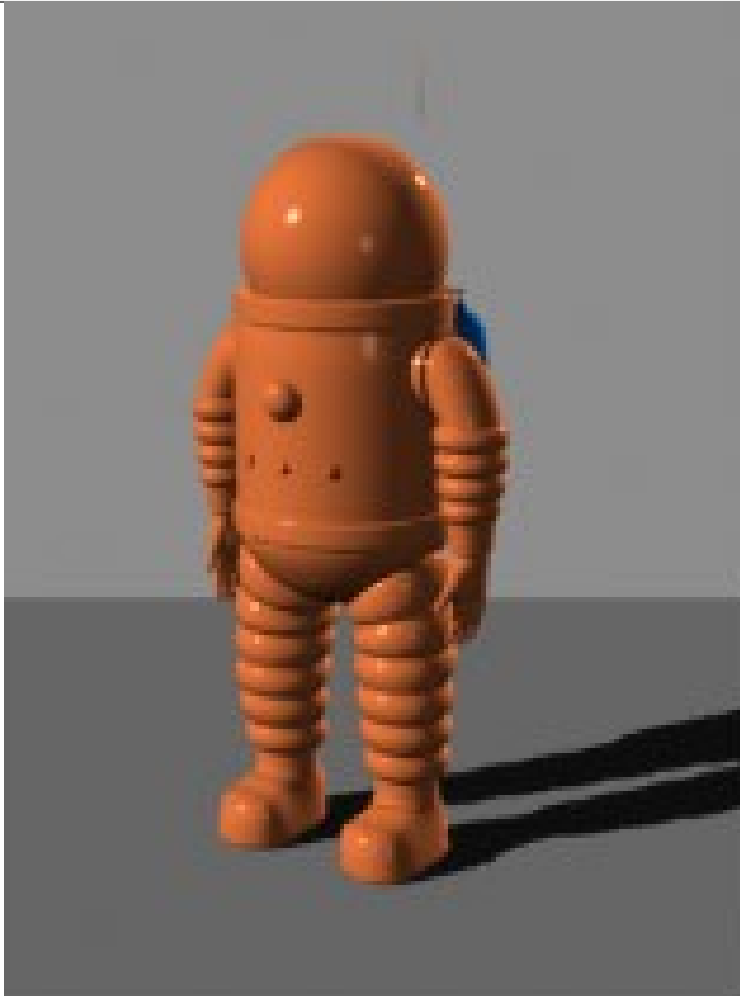
NPR - CAD - Toon

- Analogicznie postępujemy z wartością rozbłysku ($\text{dot}(H, N)$)
- Wykrywanie krawędzi:
 - Stosujemy funkcję schodkową do wyniku $\text{dot}(V, N)$
 - Powoduje błędy dla obiektów o ostrych krawędziach
 - Renderowanie dwuprzebiegowe
 - Najpierw powiększony na VS model (vertexy popchnięte w kierunku normalnych) renderujemy na kolor krawędzi
 - Dokonujemy normalnego renderingu na niemodyfikowanym modelu
 - Dokonujemy detekcji krawędzi w przestrzeni obrazu
 - Post processing
 - Większe koszty
 - Dokładne rezultaty uwzględniające wszystkie krawędzie

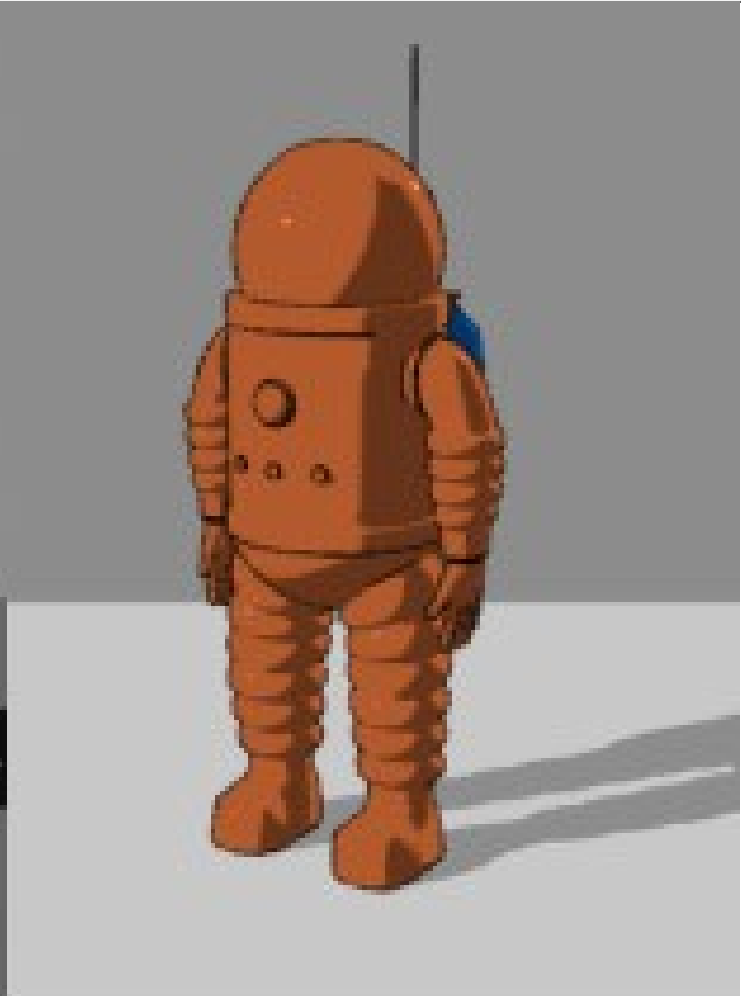
NPR - CAD - Toon



NPR - CAD - Toon



plastic shader



toon shader

NPR - CAD - Ink & Paint

- Model post processingowy
 - Udaje ilustrowanie przy pomocy tuszu i farby
 - Istnieje możliwość 'wylania się' koloru poza obrys
 - Obrys zarówno jak i farba mogą mieć teksturę udająca odręczny rysunek, pędzel itp..
 - Obrys oraz farba mogą być animowane udając np. animację odręczną





NPR - CAD - Crosshatch

- Technika udająca tradycyjne metody cieniowania
- Obszary zacięniowane ($\text{dot}(N,L) < 0$) pokrywa teksturą imitującą np. cieniowanie ołówkiem
- Tekstura może być animowana w celu symulacji animacji odręcznej
- Może być stosowana podczas renderingu obiektu jak i w post processingu

NPR - CAD - Crosshatch



NPR - CAD - Crosshatch



Pytania?



Dziękuję za uwagę

Michał Drobot
Drobot.org