Cel (Toon) shading

T120B167 Žaidimų grafinių specialiųjų efektų kūrimas ir programavimas

Rytis Maskeliūnas
Skype: rytmask
Rytis.Maskeliunas@ktu.lt

Paskaitos tema

MULTIMEDIJOS INŽINERIJOS KATEDRA

http://www.youtube.com/watch?v=TfxTehFHLy4

Ne fotorealistinis renderiavimas...

MULTIMEDIJOS INŽINERIJOS KATEDRA

- „Photorealism: age-old goal of graphic“ arba ne...
- Kas yra tas nefotorealistinis renderiavimas (angl. Non photo realistic rendering) trumpiau NFR?
  - Tai bet kokios renderiavimo technologijos, kurių tikslas nėra atkurti kuo realistiškesnį vaizdą, bet išreikšti stilių, abstrakciją, nežinomybę, emocijas, etc.
    - Dar vadinama: stylized rendering, artistic rendering, abstract rendering

NFR - Kam to reikia?

MULTIMEDIJOS INŽINERIJOS KATEDRA

- Išraiška
- Komunikacijos
  - Emocijos
  - Nuotaika
  - Menas
- Lankstumas
  - Galima keisti stilius keičiant renderiavimo metodą.

Originalas:

NPR:

http://mrl.nyu.edu/projects/npr/painterly/

NFR pvz.

- Kiekvienas renderis - savitas.
- Viršutiniai du paveikslai atitinka galutinį projektuojamą rezultatą
- Apatiniai du paveikslai gali puikiai atlikti eskizų funkciją

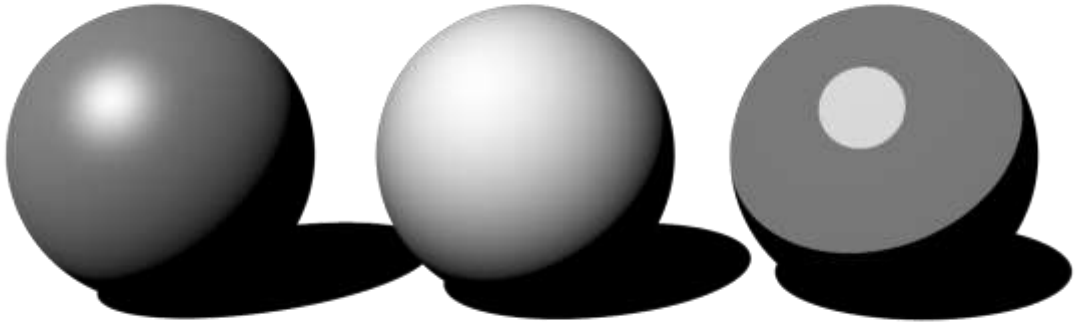
FIGURE 6.9 Different stylistic variations achieved by using different drawing tools: photorealistic rendition (a), added environment (b), pen-and-ink style (c), and painted style (d).

Pagal:
[Strothotte T., Schlechtweg, S. 2002]

Cel Shading

MULTIMEDIJOS INŽINERIJOS KATEDRA

- Dar vadinamas (Car)toon Shading arba Hard Shading.
- Pavadinimas kilo nuo spalvinimo knygelės principo. Pieštuku nupaišomi kontūrai, kurių elementus (celes) reikia nuspalvinti tam tikra spalva.
- 3D objektai atrodo kaip 2D piešinys.
- Du esminiai žingsniai: Šeidinimas (Shading), kontūrų linijų piešimas (Outline Drawing)



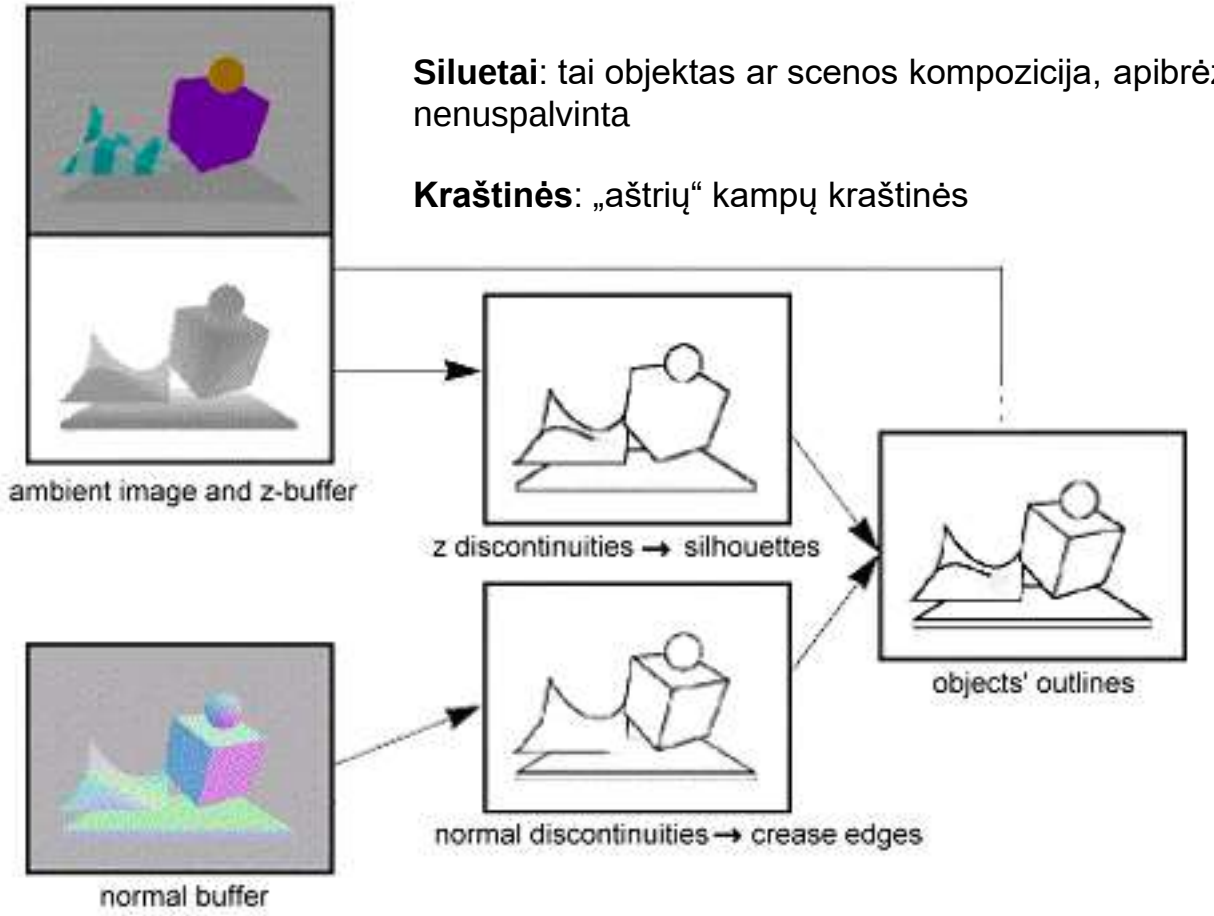
Cel shading

- Iš principo laikomasi animacinių filmų / komiksų stiliaus:
  - Paprastas, vienspalvis elementų spalvinimas (cel shading)
    - Dviejų (light/shadow) ar trijų tonų (light/shadow/highlight) paletė
  - Kraštų išryškinimas
    - Ribos (Boundary (border edge))
    - Kampai (Crease (hard edge))
    - Medžiagos kraštai (Material edge)
    - Siluetas (Silhouette edge)

Kraštų išryškinimas

- Medžiagos kraštai
  - Poligonai su bendromis kraštinėmis gali būti padengti skirtingomis medžiagomis ar tekstūrų žemėlapiais
  - Ar tai tiesiog gali būti kraštinė kurią dizaineris nori išryškinti
- Siluetas
  - Poligonai su bendromis kraštinėmis „Žiūri“ skirtingomis kryptimis (link/nuo kameros)

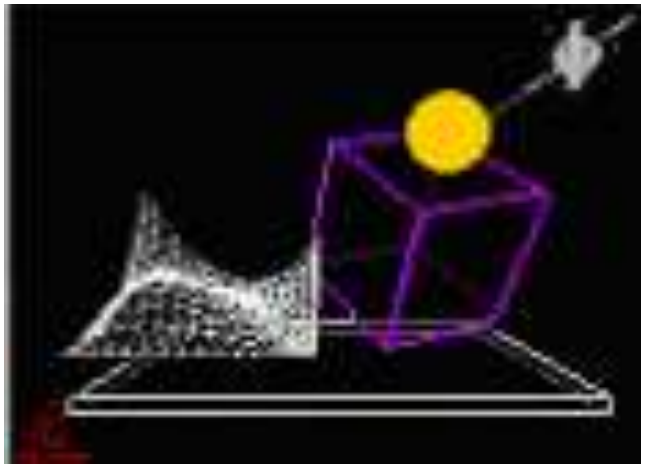
Kontūrai



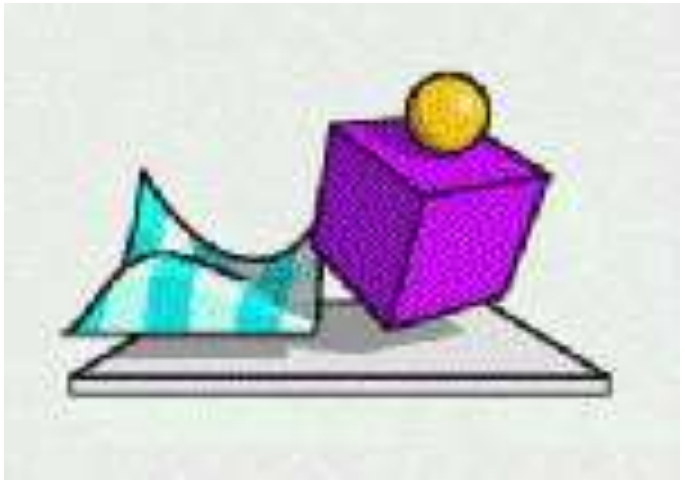
Siluetai: tai objektas ar scenos kompozicija, apibrėžta kontūro linija bet nenuspalvinta

Kraštinės: „aštrių“ kampų kraštinės

Du žingsniai



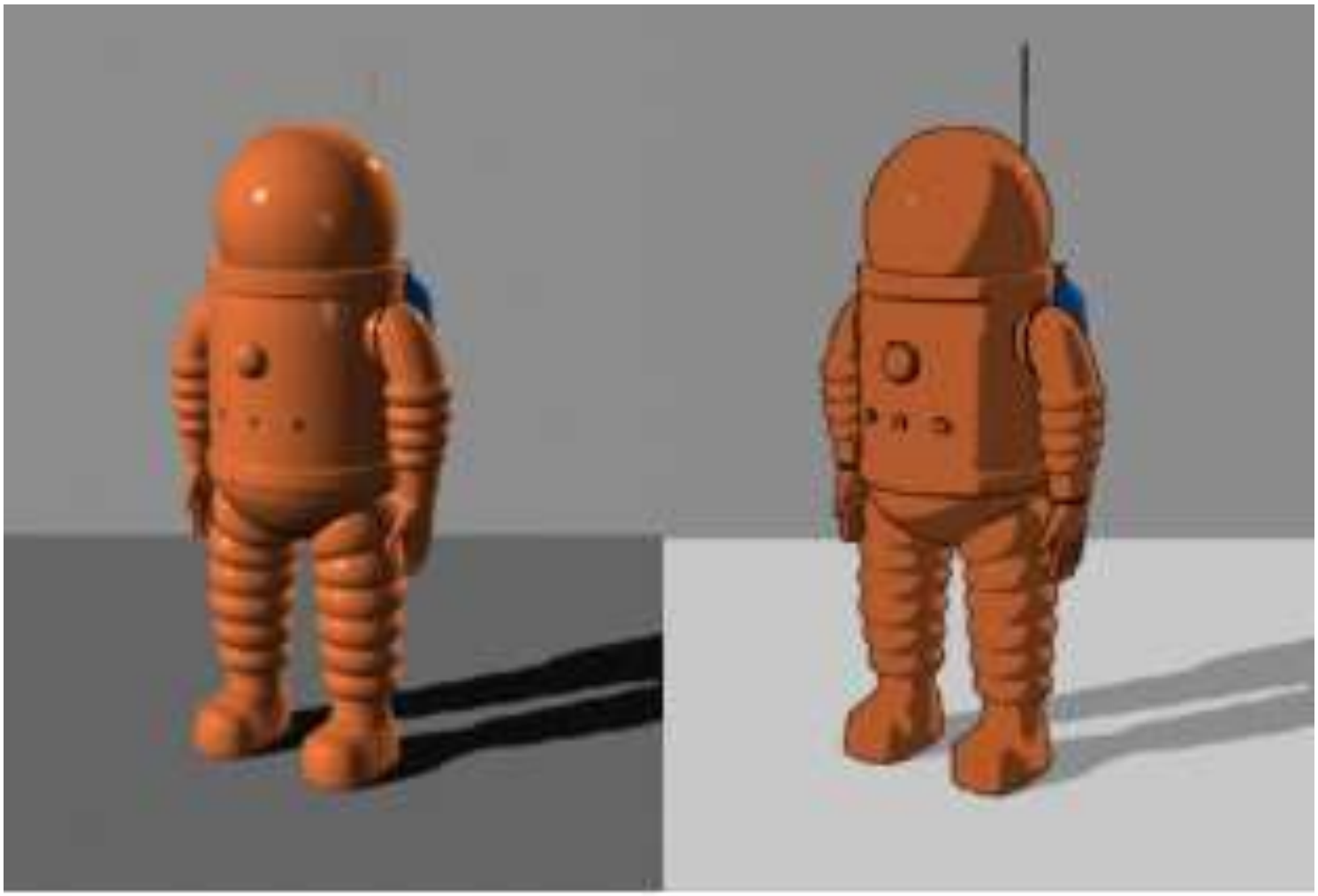
Nupaišote sceną



Renderiuojate su Cel Shading

- 2 žingsniai:
  - Nupaišote kontūrus
  - Sumažinate spalvų kiekį ☺

Skirtumas akivaizdus

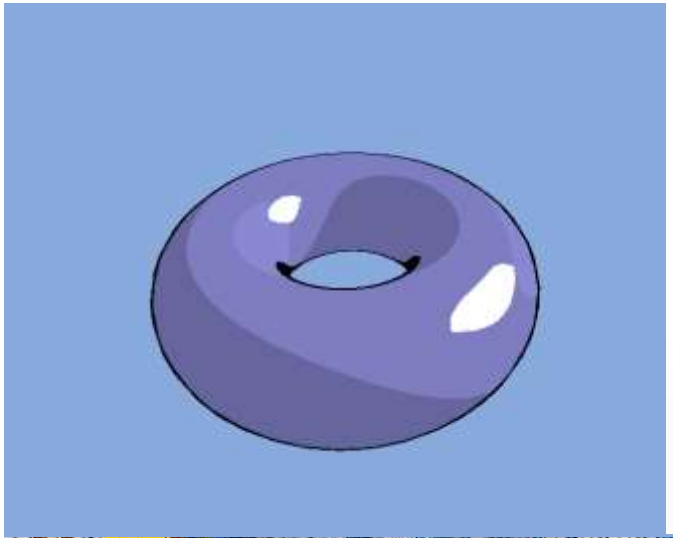


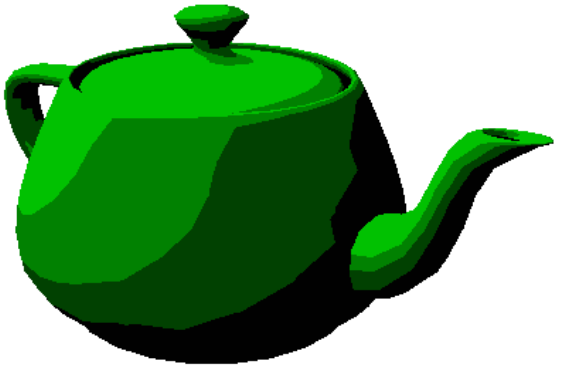
plastic shader

toon shader

Pavyzdžiai

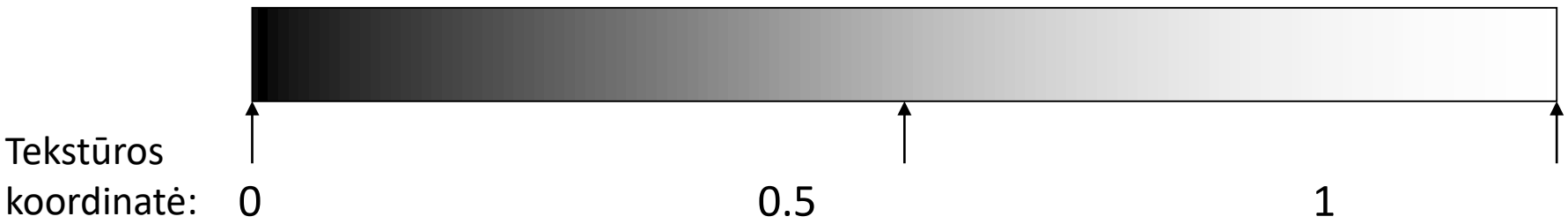
MULTIMEDIJOS INŽINERIJOS KATEDRA



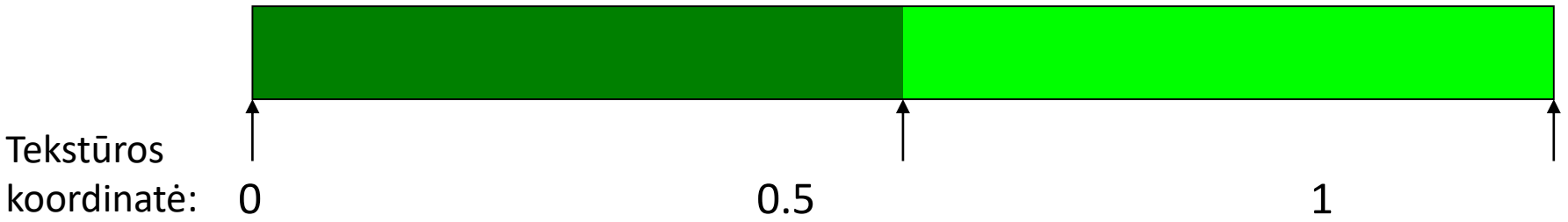


Šeidinimas

- 1D Tekstūra – tai 1x*n* tekstūra



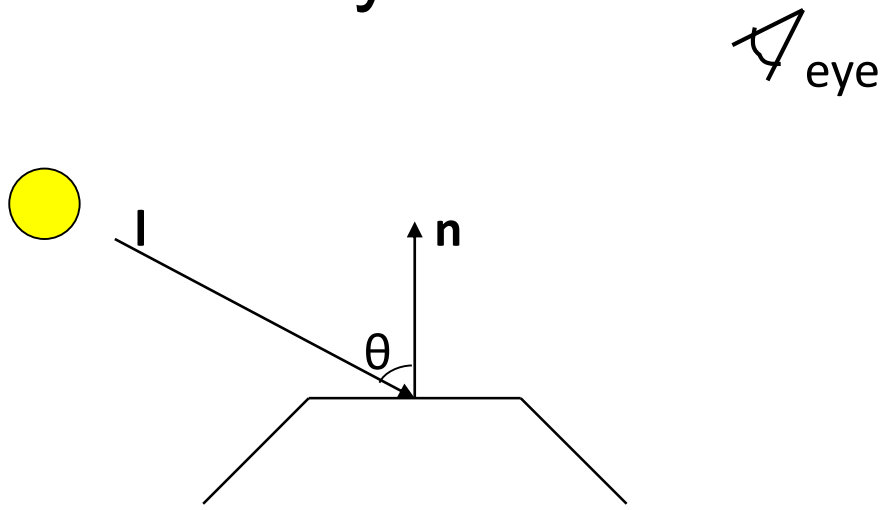
- Gali būti taip:



Tekstūros koordinatės parinkimas

MULTIMEDIJOS INŽINERIJOS KATEDRA

- Tekstūros koordinatės parinkimui naudojama šviesos lygybė.
- Galioja tradicinės taisyklės:

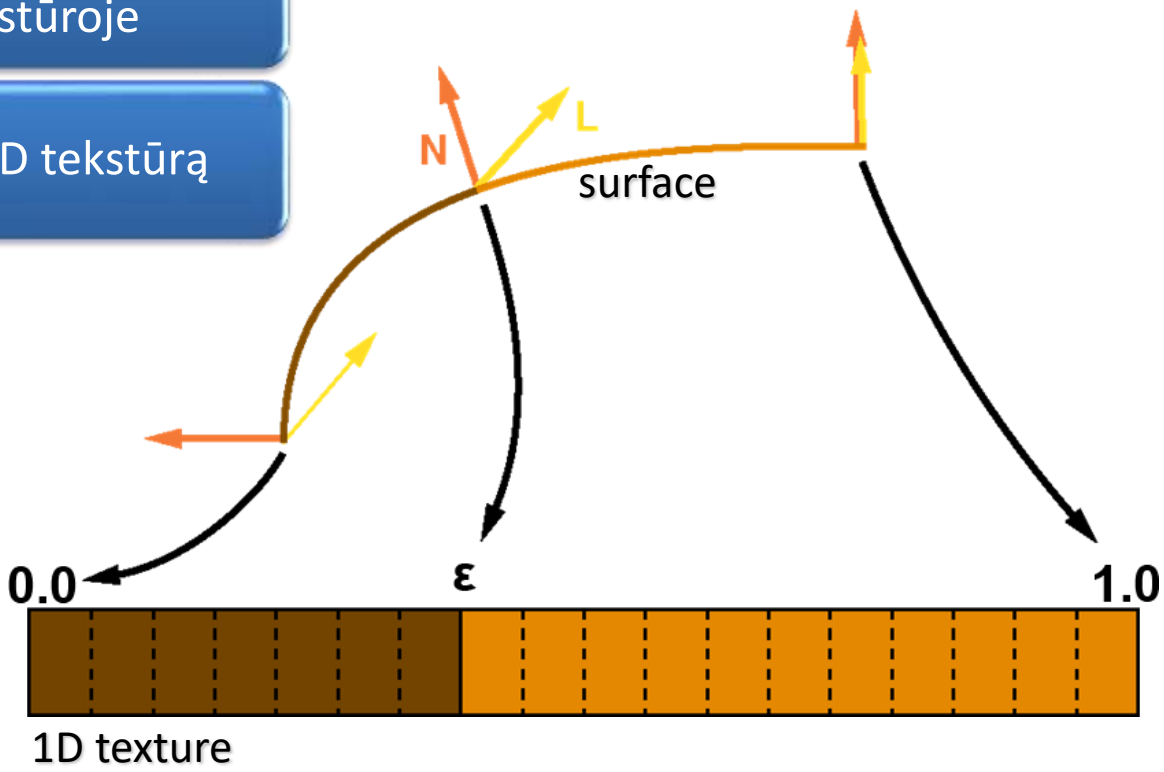


Tekstūros koordinatės parinkimas

MULTIMEDIJOS INŽINERIJOS KATEDRA

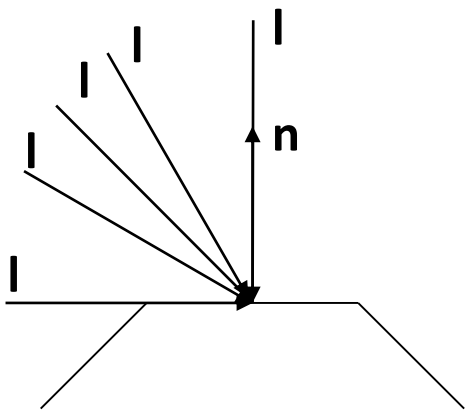
„Spalva“ objekto erdvei paskaičiuojama taip:

- Suskaičiuojam 2-3 kiekvienos spalvos atspalvius (tamsiau – šviesiau) ir saugom 1D tekstūroje
- Objektui priskiriam spalvas pagal tą 1D tekstūrą

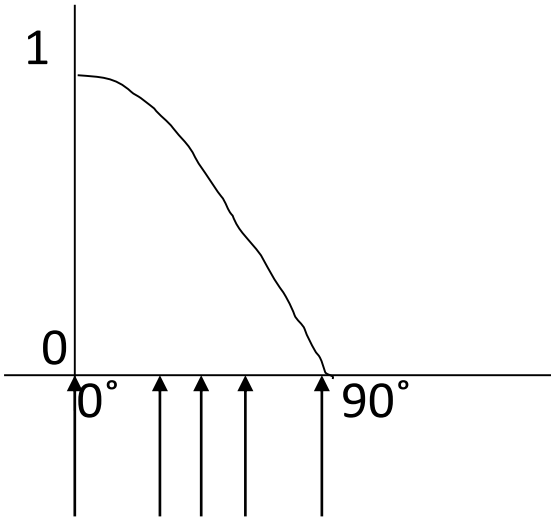


Tekstūros koordinatės parinkimas

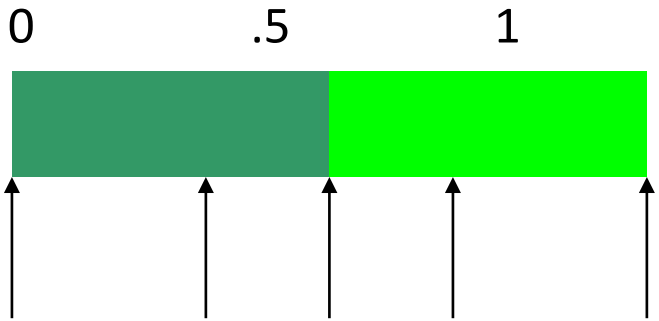
Šviesa



Cosθ



Tekstūra

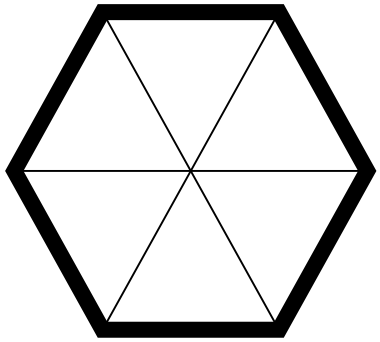


Kontūrų paišymas

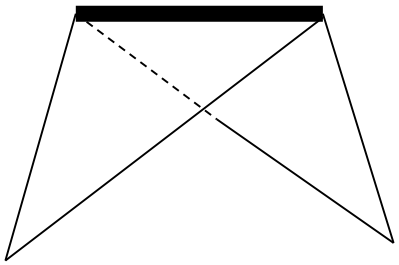
- Kuomet kraštinė yra objekto kontūro dalis?

1. Briauna – kraštine nesidalinama. Pvz., popieriaus lapo kraštai.
2. Siluetas – dalinamasi kraštine tarp priekio pusės (front-facing) ir galinės pusės (back-facing) poligonų
3. Kieti kontūrai (crease) – dalinamasi tų poligonų kraštinėmis, kurie kertasi pagal tokias pat kampo ribas (paprastai 60°).

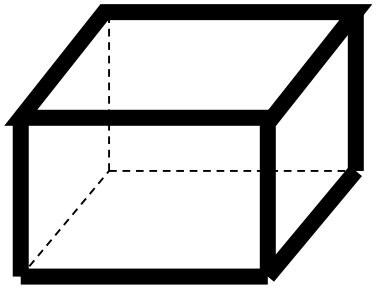
Briauna (Border)



Siluetas



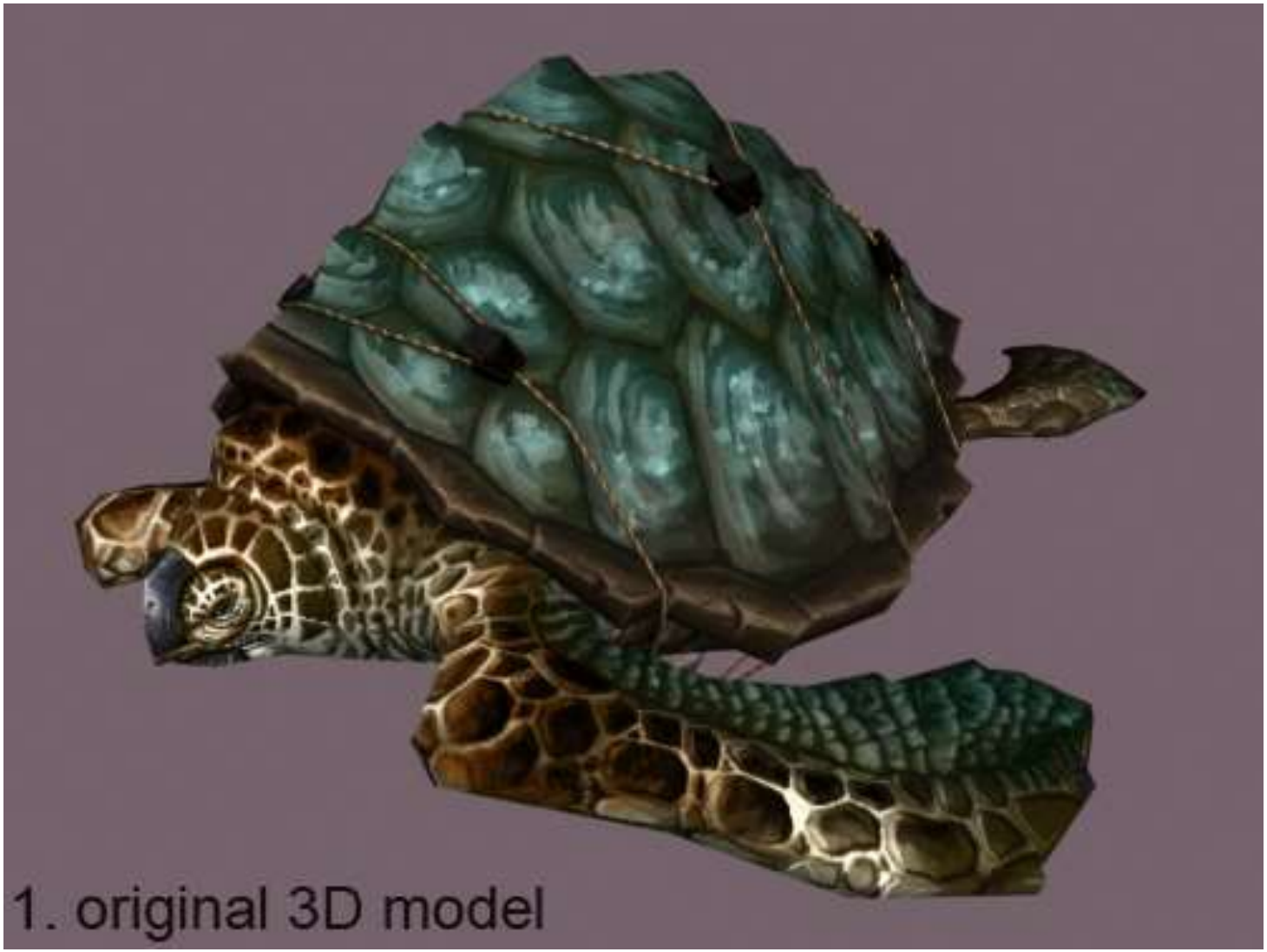
Kieti kontūrai (Hard Edges)



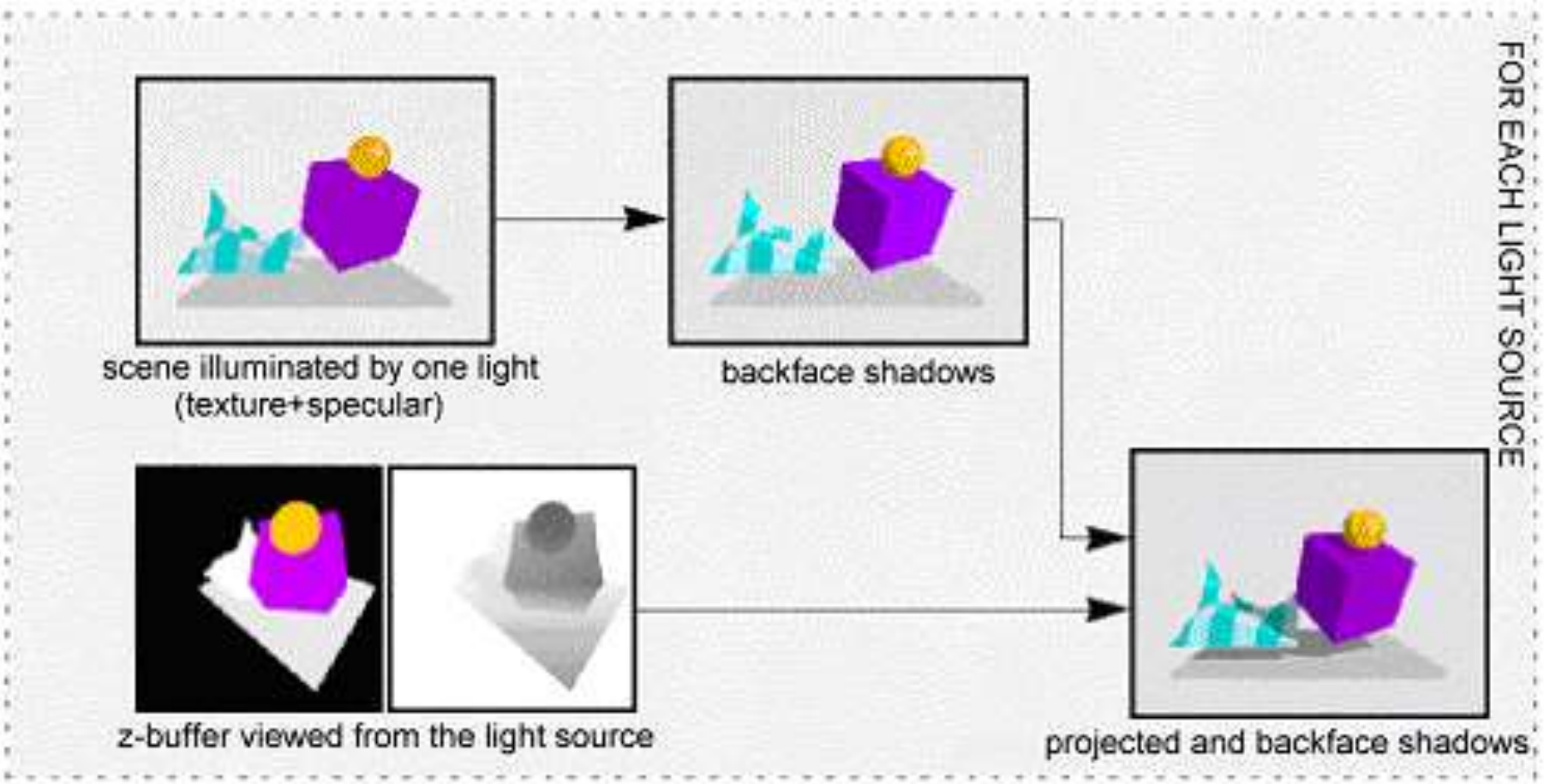
Kontūrų paišymas

- Šeideryje iš principo:
  - Skaičiuojate N●L, ir priklausomai nuo rezultato spalvinate juodai arba baltai
  - Gali būti problemų su sudėtingais paviršiais
- Reikia renderiuoti geometriją taip, kad išryškinti „siluetą“
  - Piešiam poligonus žiūrinčius į priekį (teigiama normalė)
  - Piešiam poligonus žiūrinčius atgal (neigiama normalė)
    - Galima piešti storesniu (2-pixel) wireframe
    - Arba wireframe išnešti į priekį (z-biased forward)
    - Arba pastorinti
    - Arba iškreipti per normales (“halo” efektas)

Kontūrų paišymas



Spalvinimas daromas taip



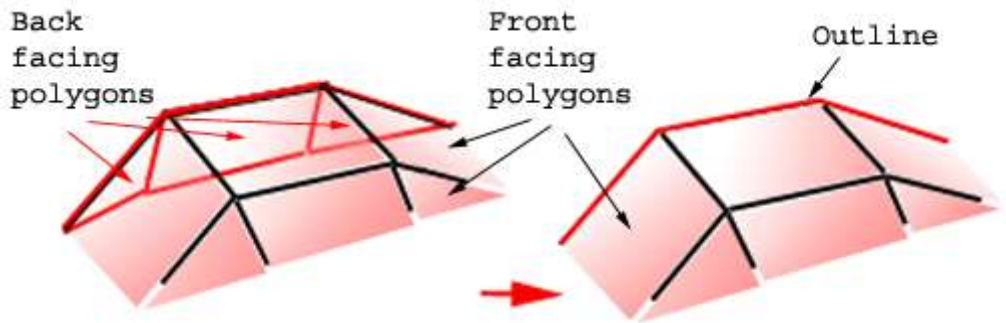
Algoritmai

- Software: (šeidinimas po nupaišymo)

Build an edge list.
for each edge in edge list
if edge not shared **then**
draw edge.
if edge belongs to front-facing and back-facing polygon **then**
draw edge
if edge belongs to two front-facing polygons that meet within some threshold **then**
draw edge

- Hardware: (šeidinimas po nupaišymo)

Naudojamas Z-Buffer (gylio koordinatės) ir HW palaikantis priekinės ir galinės pusės išdėstymo (teigiama/negiama normalės kryptis) poligonų paišymą (hardware support for drawing front-facing or back-facing polygons).

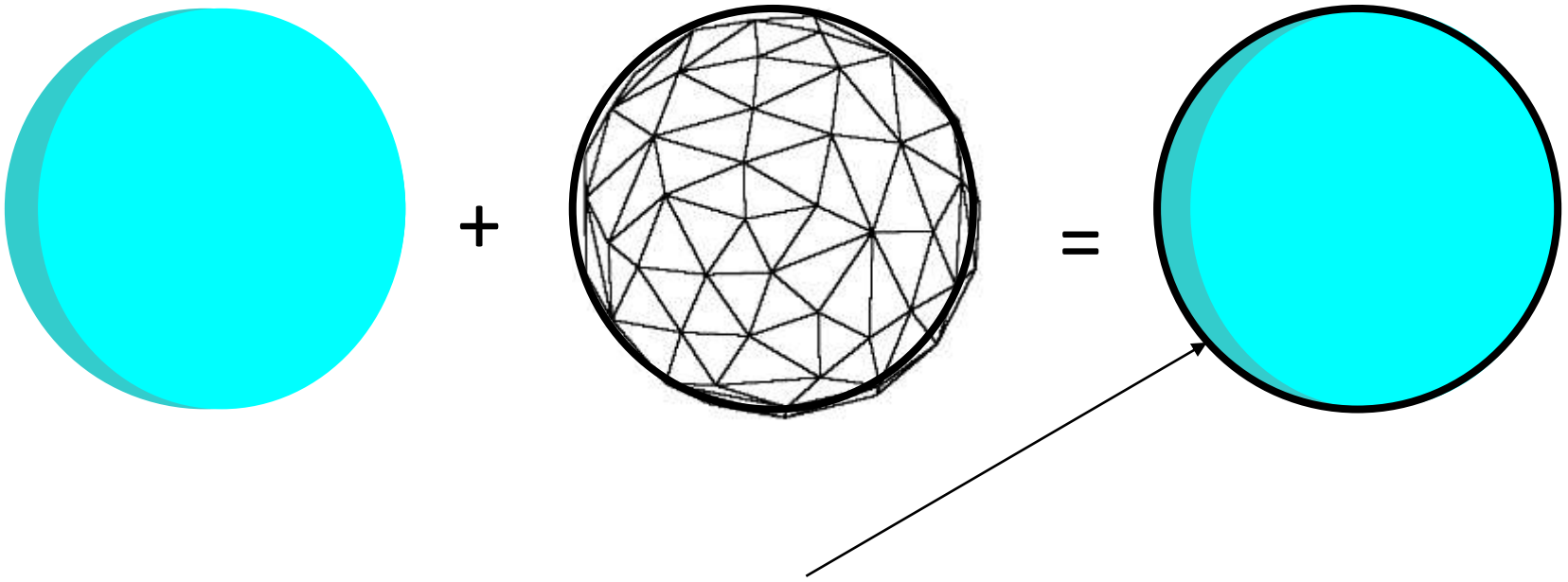


Kaip veikia HW

Šeidinimas

Sferos tinklelio galines pusės išdėstymo poligonai (kontūrai)

Gauna, cell-shaded sferą



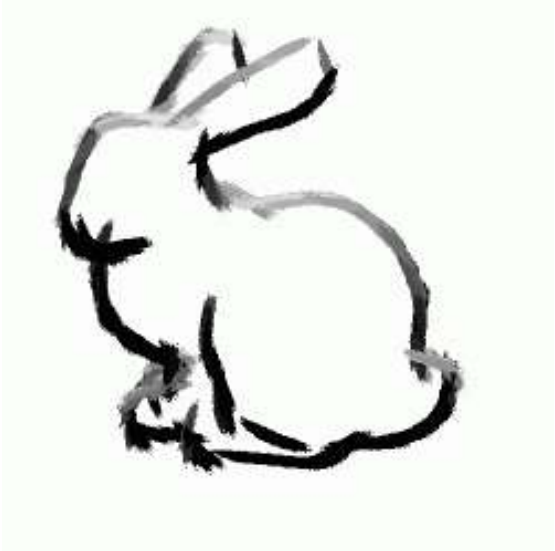
Kontūrą sugeneruoja Z-bufferis. Paišoma tik tie galinės pusės išdėstymo krypties tinklelio (back-facing wireframe) kraštai kurie yra tokio pačio gylio (depth) kaip ir šeideriuoti priekinės pusės krypties išdėstymo poligonai (t.y., silueto kraštai).

HW vs SW

Software	Hardware
Reikia saugoti kraštinių sąrašą	Nėra papildomų duomenų struktūrų
Kontūrą galima nupiešti tuo pačiu renderiavimo etapu kaip ir šeidinimas	Kontūrą reikia nupiešti papildomu renderiavimo etapu
Palaiko briaunas, kontūrus ir kietas kraštines (borders, outlines, hard edges).	Paišo tik siluetus ir kai kurias briaunas.
Sunkiau implementuoti.	Labai lengva implementuoti.

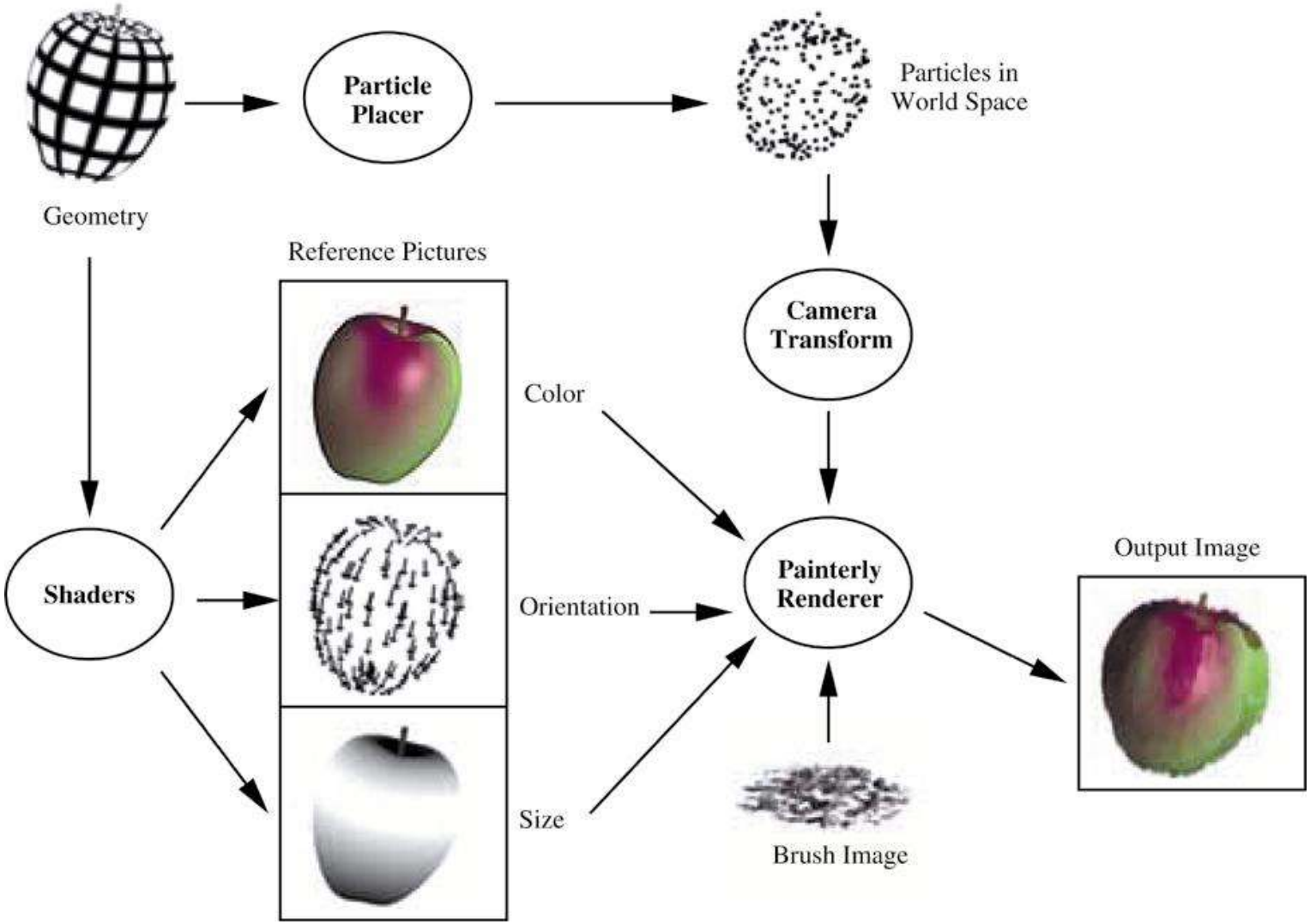
Galimi ir kitokie stiliai

- Impresionistinis ar „teptuko“ renderiavimas:
  - Galima „užpurkšti“ daleles ant objekto paviršius
  - Galima piešti „teptuko“ potėpiais
  - Galima užkoduoti normales, paviršiaus kreives, gylį, spalvos/tono informaciją ir kt.



Teptuko renderiavimas (Painterly Rendering)

MULTIMEDIJOS INŽINERIJOS KATEDRA



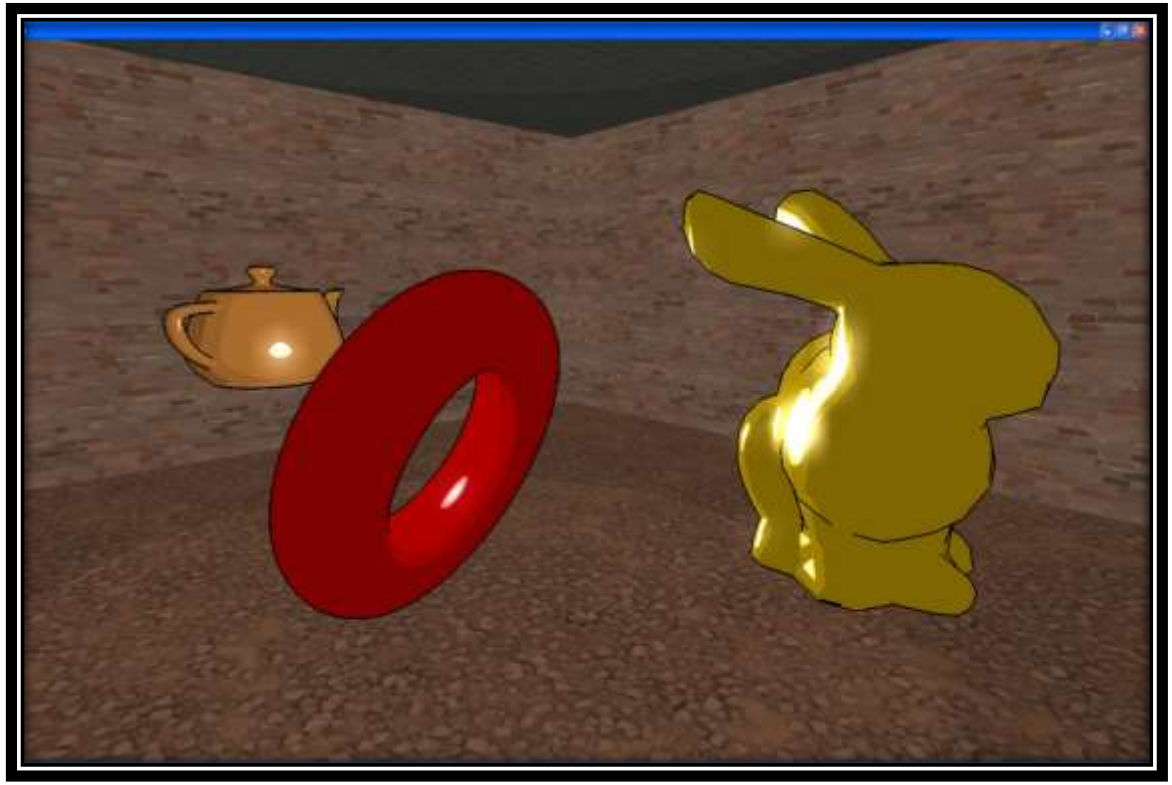
Galima maišyti stilius

MULTIMEDIJOS INŽINERIJOS KATEDRA

VS

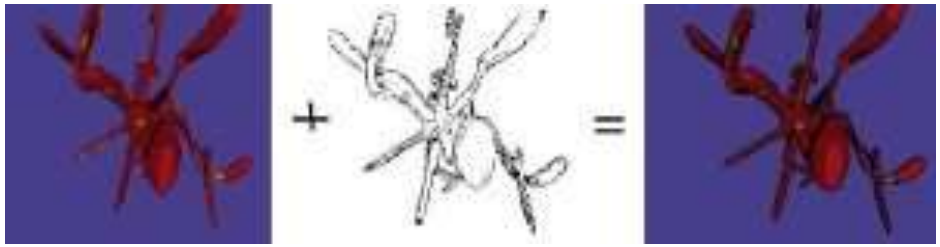
Cell shading

- Norint sukurti tokį efektą reikia realizuoti dvi funkcijas:
- (a) Reikia pridėti apšvietimą pagal tekstūrą
- (b) Reikia nustatyti kraštus (kontūrams).



Cell shading - spalva

- Taigi, reikia surenderiuoti sceną su pirmuoju šeideriu (a), tuomet panaudojus antrąjį reikia nustatyti kraštines ir apjungti į bendrą vaizdą:
- Shader (a) + (b) =



- Iš pradžių reikia suskaičiuoti objekto difuzinę šviesą (N dot L), o rezultatą naudoti kaip tekstūros x koordinatę:

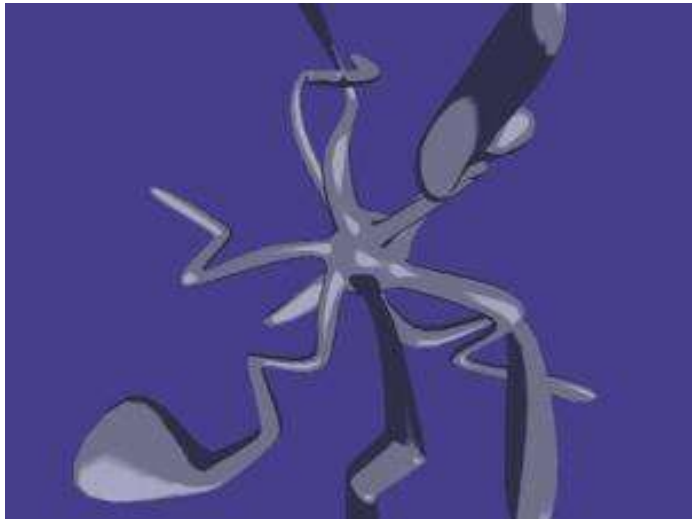
```
Tex.y = 0.0f;
Tex.x = saturate(dot(L, N));

float4 CelColor = tex2D(CelMapSampler, Tex);
```

- Čia CelMapSampler nuskaito duomenis iš tekstūros (rezoliucija 32×1):

Cell shading - spalva

- Jeigu kampas tarp L ir N yra didelis (dot rezultatas = 0) naudojama tekstūros koordinatė 0.0,0.0.
- Jei kampas tarp N ir L yra lygus 0(dot rezultatas = 1) naudojama koordinatė 1.0,0.0
- Bet kokios kitos vertės bus tarp šio intervalo (0,0 ... 1,0)
- Realiai turite tik 3 spalvas.
- Gražinant CelColor iš pikselių šeiderio, gausite difuzinį šeidinimą pagal panaudotą tekstūrą:



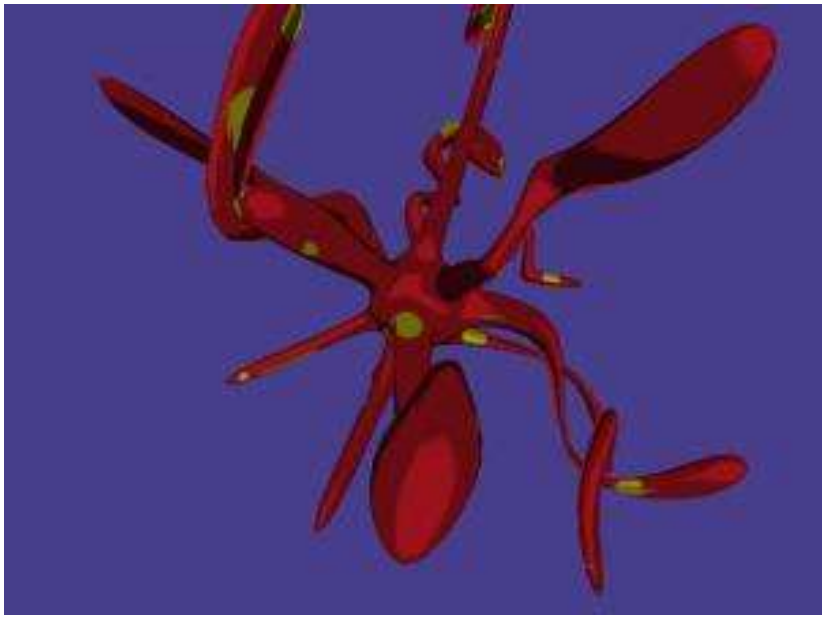
Cell shading su HLSL - spalva

MULTIMEDIJOS INŽINERIJOS KATEDRA

- Jei norime spalvotos tekstūros reikia difuzinę spalvą padauginti iš tekstūros spalvos, t.y. dauginam tekstūros spalvą su difuziniu žemėlapiu CelColor:

```
return (Ai*Ac*Color)+(Color*Di*CelColor);
```

- Rezultate bus taip:



Cell shading su HLSL - spalva


```
float4 PixelShaderGo(float2 Tex: TEXCOORD0,float3 L: TEXCOORD1, float3 N: TEXCOORD2) : COLOR
{
	// Calculate normal diffuse light but use Tex.x as color in stead.
	float4 Color = tex2D(ColorMapSampler, Tex);
	float Ai = 0.8f;
	float4 Ac = float4(0.075, 0.075, 0.2, 1.0);
	float Di = 1.0f;
	float4 Dc = float4(1.0, 1.0, 1.0, 1.0);

	Tex.y = 0.0f;
	Tex.x = saturate(dot(L, N));

	float4 CelColor = tex2D(CelMapSampler, Tex);

	return (Ai*Ac*Color)+(Color*Di*CelColor);
}
```

Cell shading - Kontūrai

- Kontūrus galima pasidaryti dviem būdais:
- Surenderiuoti išverstą (culled) objektą visiškai juodą ir tuomet nurenderiuoti cel-shading apdorotą versiją, tačiau kiek mažesnio dydžio;
- Galima renderiuoti sceną į tekstūrą ir pasigaminti kraštų aptikimo šeiderį (ką ir darysime).

Cell shading - Kontūrai

- Kraštinių aptikimo šeideris gali būti realizuotas kernelio filtru su tokia matrica:

-1	-2	-1
0	0	0
1	2	1

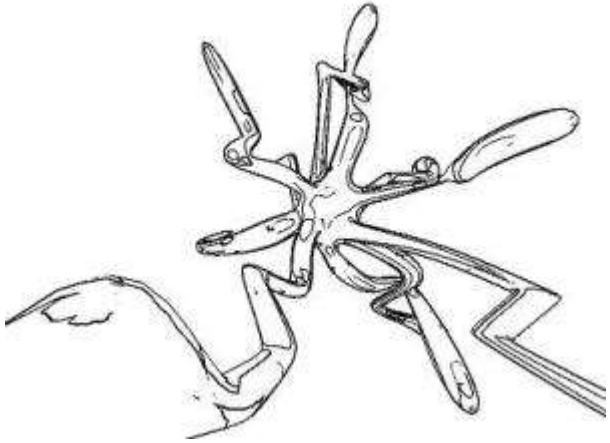
- Kernelio filtras veikia pritaikant kernelio matricą kiekvienam renderiuojamo vaizdo pikseliui. Kernelyje yra sužymėti daugybos faktoriai kuriuos reikia pritaikyti pikseliui ir jo kaimynams.
- Kai sudauginamos visos vertės, pikselis pakeičiamas rezultatų suma.
- Parenkant skirtingus kernelius galima gauti skirtingus filtravimo tipus.

Color*float4(result.xxx,1);

Cell shading - Kontūrai

MULTIMEDIJOS INŽINERIJOS KATEDRA

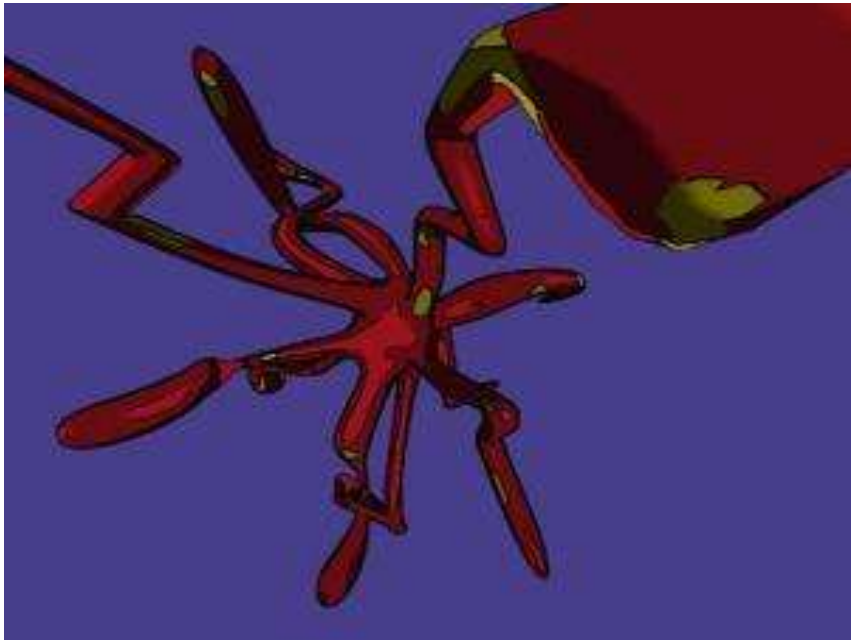
- Tokį šeiderį pritaikius tekstūrai, gausite juodai baltą tekstūrą kur kraštinės bus juodos spalvos, o visa kita bus balta



- Viskas ką jums liko padaryti tai apjungtį spalvas su šiuo paveikslu:
- Color*float4(result.xxx,1);
- Čia Color yra scenos tekstūra (scenetexture), o result.xxxx yra gautoji kraštinių tekstūrų

Cell shading su HLSL - Kontūrai

- Visi pikseliai kurie yra ne kraštinės dalis bus balti (vertė 1.0)
- Visi kraštinės pikseliai bus juodi (0.0)
- Sudauginus spalvą (Color) iš 1.0 toje vietoje gaunate spalvotą vaizdą, sudauginus iš 0, gaunate 0.0 ir atitinkamai juodą spalvą (kontūrą).



Cell shading su HLSL - Kontūrai

MULTIMEDIJOS INŽINERIJOS KATEDRA

```
float4 PixelShaderGo(float2 Tex: TEXCOORD0) : COLOR
{
        float4 Color = tex2D(ColorMapSampler, Tex);
        float2 QuadScreenSize = float2(800,600);

        float2 ox = float2(Thickness/QuadScreenSize.x,0.0);
   float2 oy = float2(0.0,Thickness/QuadScreenSize.y);
   float2 uv = Tex.xy;
   float2 PP = uv - oy;
   float4 CC = tex2D(ColorMapSampler,PP-ox); float g00 = getGray(CC);
   CC = tex2D(ColorMapSampler,PP);    float g01 = getGray(CC);
   CC = tex2D(ColorMapSampler,PP+ox); float g02 = getGray(CC);
   PP = uv;
   CC = tex2D(ColorMapSampler,PP-ox); float g10 = getGray(CC);
   CC = tex2D(ColorMapSampler,PP);    float g11 = getGray(CC);
   CC = tex2D(ColorMapSampler,PP+ox); float g12 = getGray(CC);
   PP = uv + oy;
   CC = tex2D(ColorMapSampler,PP-ox); float g20 = getGray(CC);
   CC = tex2D(ColorMapSampler,PP);    float g21 = getGray(CC);
   CC = tex2D(ColorMapSampler,PP+ox); float g22 = getGray(CC);
   float K00 = -1;
   float K01 = -2;
   float K02 = -1;
   float K10 = 0;
   float K11 = 0;
   float K12 = 0;
   float K20 = 1;
   float K21 = 2;
   float K22 = 1;
   float sx = 0;
   float sy = 0;
   sx += g00 * K00;
   sx += g01 * K01;
   sx += g02 * K02;
   sx += g10 * K10;
   sx += g11 * K11;
   sx += g12 * K12;
   sx += g20 * K20;
   sx += g21 * K21;
   sx += g22 * K22;
   sy += g00 * K00;
   sy += g01 * K10;
   sy += g02 * K20;
   sy += g10 * K01;
   sy += g11 * K11;
   sy += g12 * K21;
   sy += g20 * K02;
   sy += g21 * K12;
   sy += g22 * K22;
   float dist = sqrt(sx*sx+sy*sy);
   float result = 1;
   if (dist>Threshold) { result = 0; }

   // The scene will be in black and white, so to render
   // everything normaly, except for the edges, bultiply the
   // edge texture with the scenecolor
   return Color*result.xxxx;

}
```

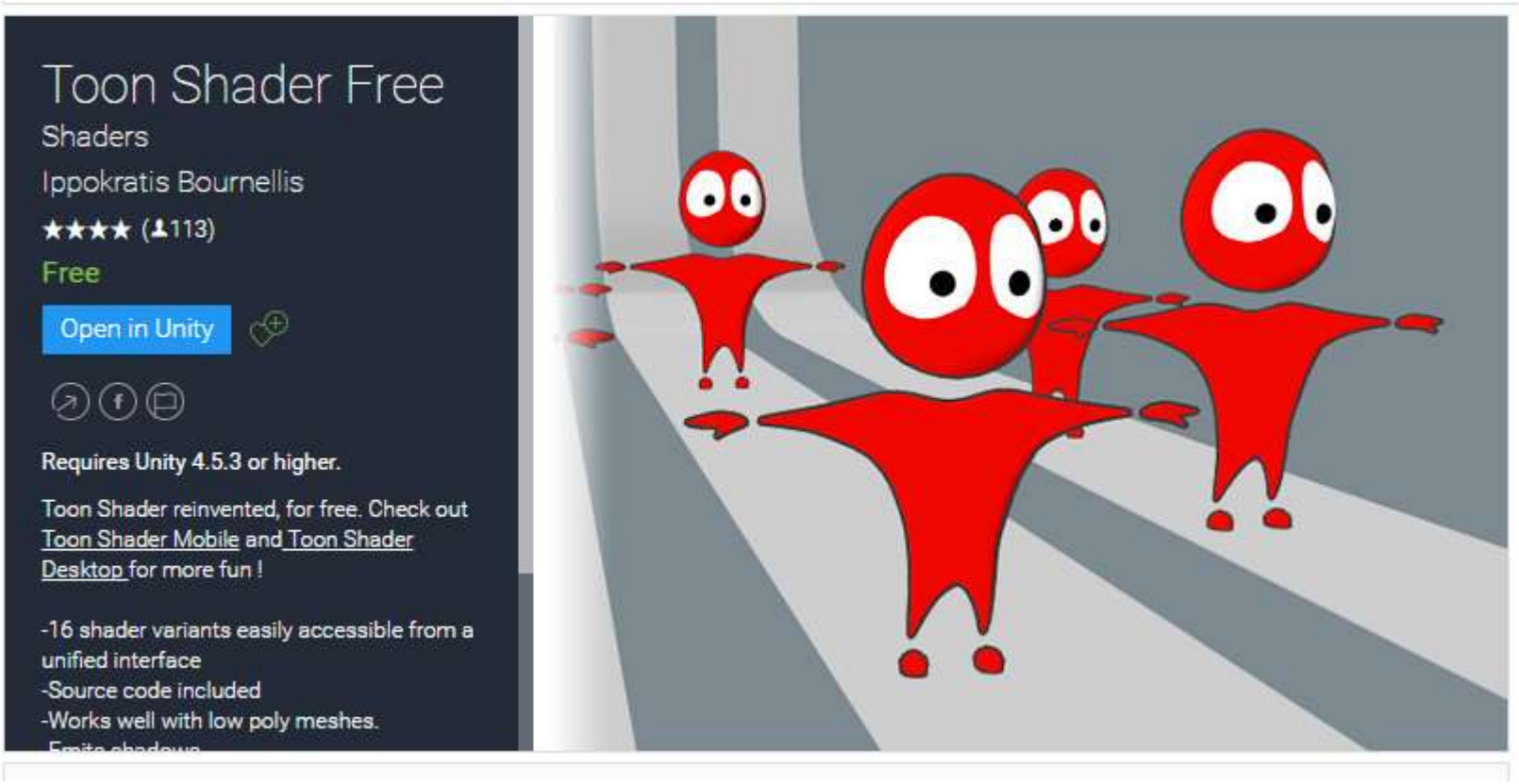
Unity3D toon shading

MULTIMEDIJOS INŽINERIJOS KATEDRA

- a) pavyzdys moodle;
- b) Sudėtingesni Effects pakete.

Rinkinys idėjoms

MULTIMEDIJOS INŽINERIJOS KATEDRA



https://www.assetstore.unity3d.com/en/#!/content/21288