

GLSL
Textura Procedural

Waldemar Celes

Tecgraf/PUC-Rio





Antialiasing



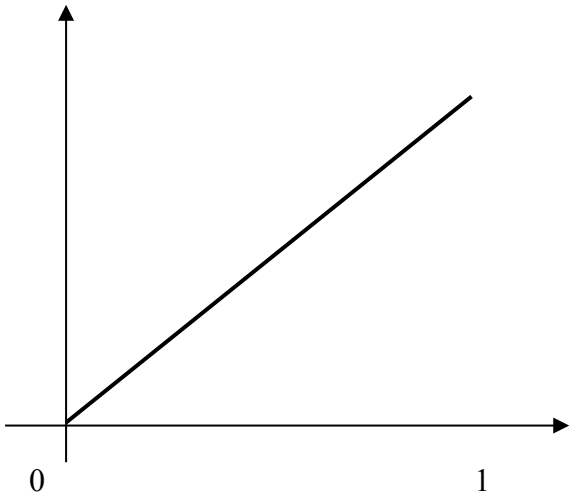
Antialiasing: textura procedural

- Exemplo
  - ⇨ Esfera com faixas (strips)

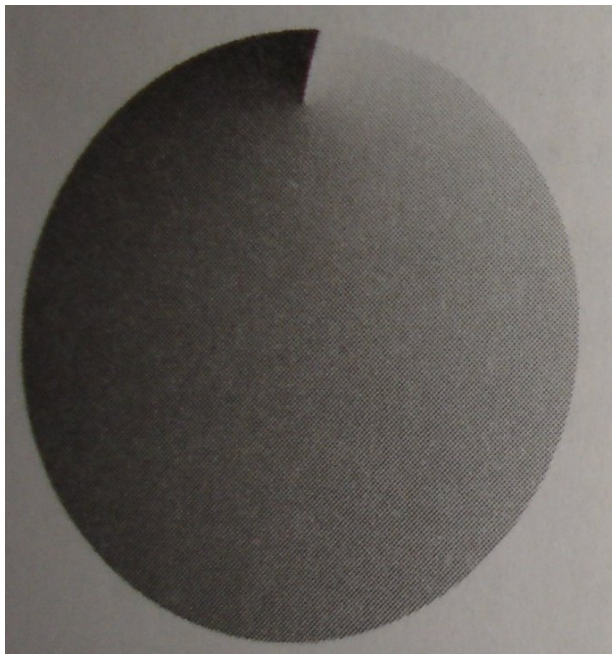


Antialiasing: textura procedural

- Como obter textura de faixas?
  - ⇨ 1º passo: luminosidade dada por *s*



float itensity = S

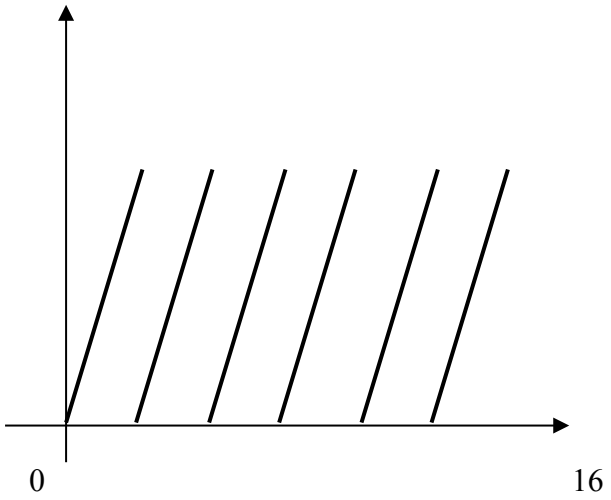


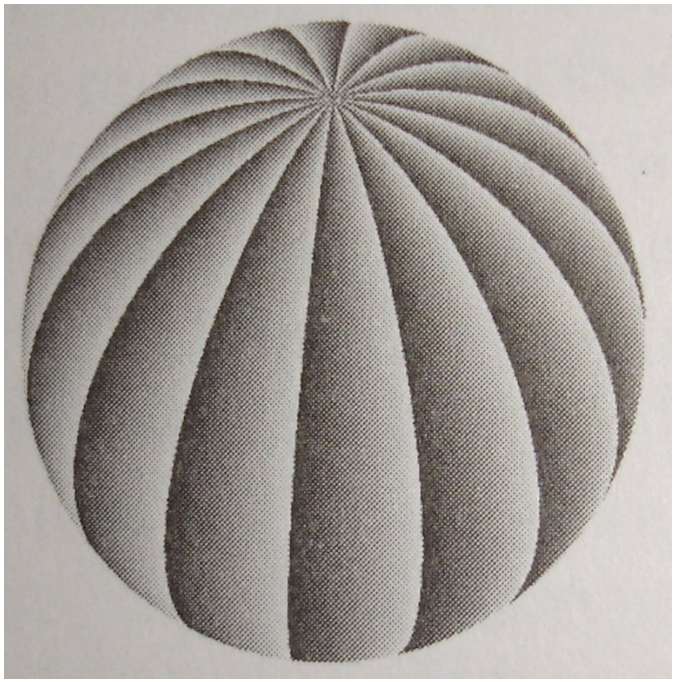
TecGraf PUC RIO

Antialiasing: textura procedural

- Como obter textura de faixas?
  - 2º passo: luminosidade variável (*sawtooth*)
    - Considerando 16 faixas

float sawtooth = fract (S * 16)



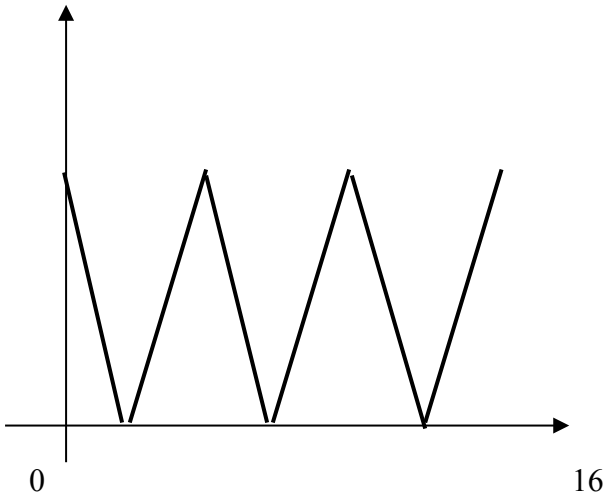


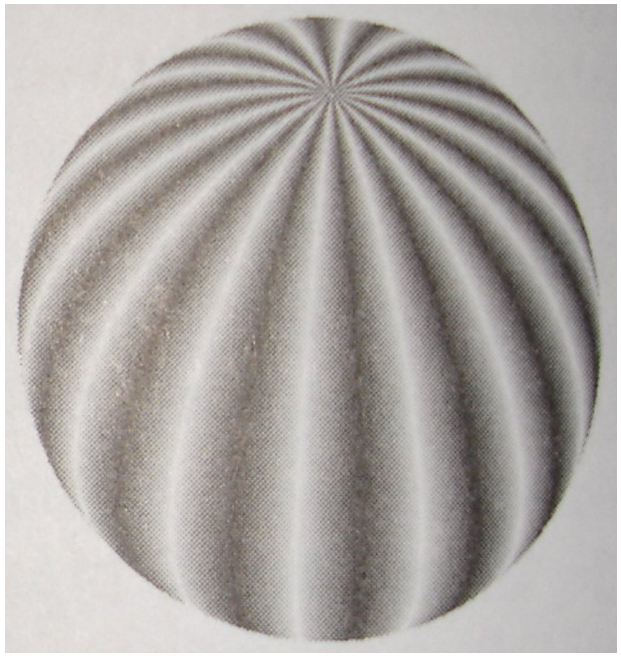
TecGraf PUC RIO

Antialiasing: textura procedural

- Como obter textura de faixas?
  - ⇨ 3º passo: luminosidade variável (*abs*)

float triangle = abs(2.0*sawtooth – 1.0)

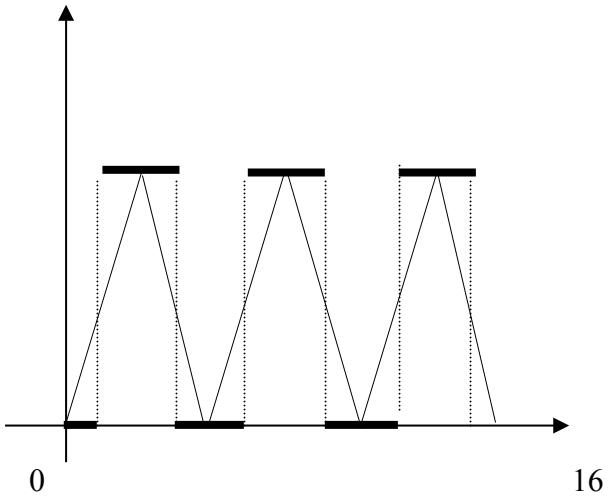




TecGraf PUC RIO

Antialiasing: textura procedural

- Como obter textura de faixas?
  - ⇨ 4º passo: luminosidade em faixa (*step*)



float square = step (0.5, triangle)



TecGraf PUC RIO

Antialiasing: textura procedural

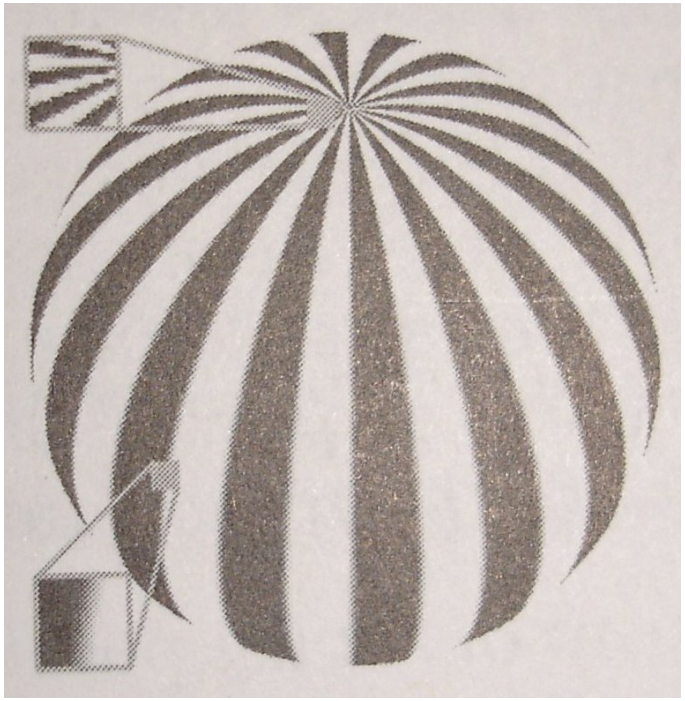
- Aliasing, como eliminar?



Antialiasing: textura procedural

- Transição suave (*smoothstep*)
  - ⇨ Eliminação das freqüências altas
    - ✧ Eliminação das arestas vivas

float square = smoothstep (0.4, 0.6, triangle)



TecGraf PUC RIO

Antialiasing: textura procedural

- Problemas com *smoothstep*
  - Tamanho do filtro (0.2) fixo
    - Parâmetro varia em coordenadas de tela
  - Funciona bem para linha do equador
    - Baixa taxa de variação de S
  - Funciona mal para outras regiões
    - Alta taxa de variação de S

Antialiasing: textura procedural

- Tamanho do filtro adaptativo
  - ⇨ De acordo com variação de S no espaço da tela
- Variação de S
  - ⇨ Gradiente de uma função *f*

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

- Taxa de variação de *f*
  - ⇨ Magnitude do gradiente

$$\mathbf{mag}[\nabla f] = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$

TecGraf PUC RIO

Antialiasing: textura procedural

- Funções *built-in*

$$\mathbf{dFdx} = \frac{\partial f}{\partial x}$$

$$\mathbf{dFdy} = \frac{\partial f}{\partial y}$$

$$\mathbf{fwidth} = \left|f(x,y) - f(x+1,y)\right| + \left|f(x,y) - f(x,y+1)\right| \cong \mathbf{mag}\left[\nabla f\right]$$

- Cálculo da taxa de variação

```
float width = length(vec2(dFdx(S),dFdy(S)));
```

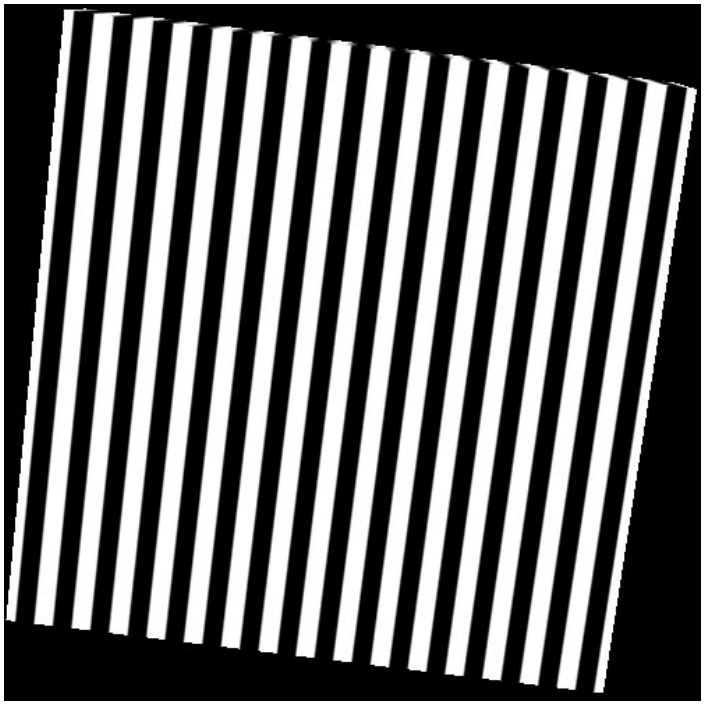
ou

```
float width = fwidth(S);
```

TecGraf PUC RIO

Antialiasing: textura procedural

- Fragment shader: filtro com tamanho adaptativo



```
in float V;                 // generic varying
in float LightIntensity;

uniform float Frequency;  // Stripe frequency = 16

void main (void)
{
    float sawtooth = fract(V * Frequency);
    float triangle = abs(2.0 * sawtooth - 1.0);
    float width = length(vec2 (dFdx(V), dFdy(V)));
    float edge = width * Frequency * 2.0;       // 2 pixels large
    float square = smoothstep(0.5 - edge, 0.5 + edge, triangle);
    gl_FragColor = vec4 (vec3 (square), 1.0) * LightIntensity;
}
```

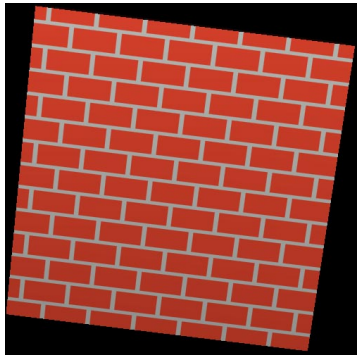
TecGraf PUC RIO

Antialiasing: textura procedural

- Problemas: filtro com tamanho adaptativo
  - Funciona bem desde que freqüência não seja muito alta
  - Região dos polos
    - Largura das faixas bem menores que um pixel
  - Problema similar com reconstrução de textura
    - Técnica de *mipmapping*
  - Solução
    - Integração do sinal
    - Limitação da freqüência

Antialiasing: textura procedural

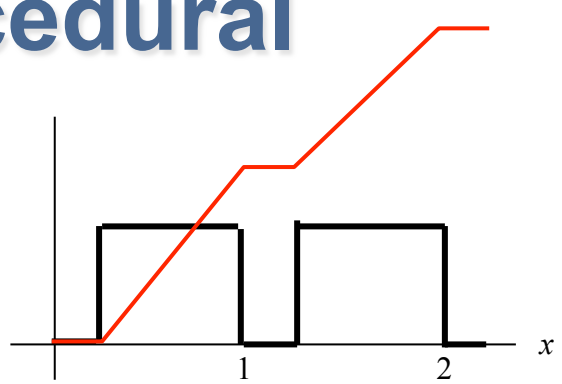
- Integração analítica
  - Exemplo da textura de tijolo
    - Função step: tijolo-cimento



  - Integração analítica
    - Avalia integral da função, em vez da função em si
  - Usa integral da função para avaliar valor médio
    - Funciona para qualquer freqüência

Antialiasing: textura procedural

- Exemplo da textura de tijolo
  - Valor da integral analítica




```
#define Integral (x, p, notp) ((floor(x)*(p) + max(fract(x)-(notp),0.0))
```

  - Uso da integral analítica
    - Obtenção do valor médio dentro da largura do filtro

```
vec2 fw, useBrick;

fw = fwdith(position);

useBrick = (Integral(position+fw, BrickPct, MoltarPct) –
                Integral(position, BrickPct, MoltarPct))  /  fw;  // box filter
```

TecGraf PUC RIO

Antialiasing: textura procedural

- Fragment shader: textura de tijolo (1/3)

```
uniform vec3  BrickColor, MortarColor;
uniform vec2  BrickSize;
uniform vec2  BrickPct;
uniform vec2  MortarPct;

in vec2  MCposition;       /* modeling coordinate position */
in float LightIntensity;

#define Integral(x, p, notp) ((floor(x)*(p)) + max(fract(x)-(notp), 0.0))
```

Antialiasing: textura procedural

- Fragment shader: textura de tijolo (2/3)

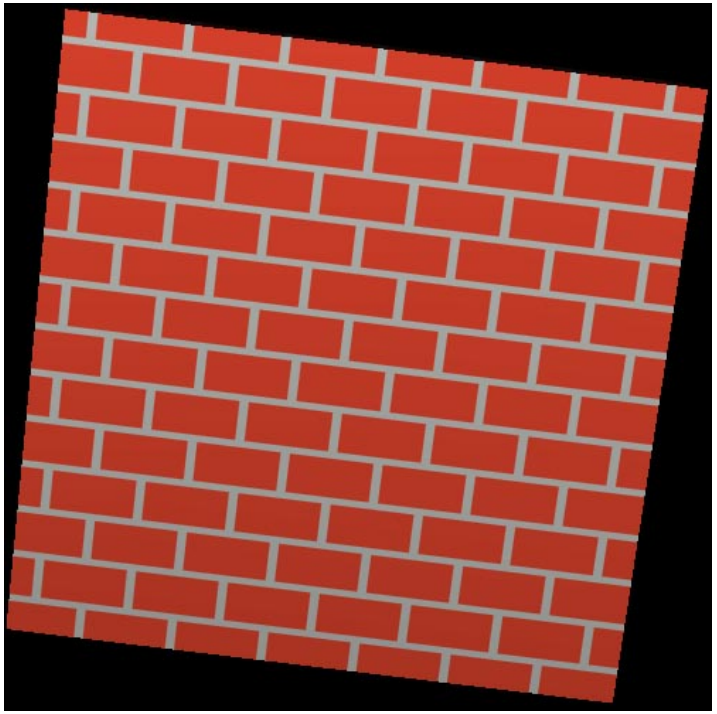
```
void main(void)
{
    vec2 position, fw, useBrick;
    vec3 color;
     // Determine position within the brick pattern (“brick number”)
    position = MCposition / BrickSize;
    // Adjust every other row by an offset of half a brick
    if (fract(position.y * 0.5) > 0.5)
        position.x += 0.5;
    // Calculate filter size
    fw = fwidth(position);
    // Perform filtering by integrating the 2D pulse
    useBrick = (Integral(position + fw, BrickPct, MortarPct) -
                Integral(position, BrickPct, MortarPct)) / fw;
```

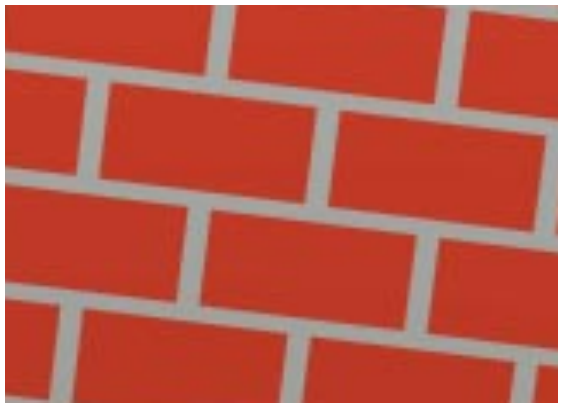
TecGraf PUC RIO

Antialiasing: textura procedural

- Fragment shader: textura de tijolo (3/3)

```
    // Determine final color
    color  = mix(MortarColor, BrickColor, useBrick.x * useBrick.y);
    color *= LightIntensity;
    gl_FragColor = vec4 (color, 1.0);
}
```





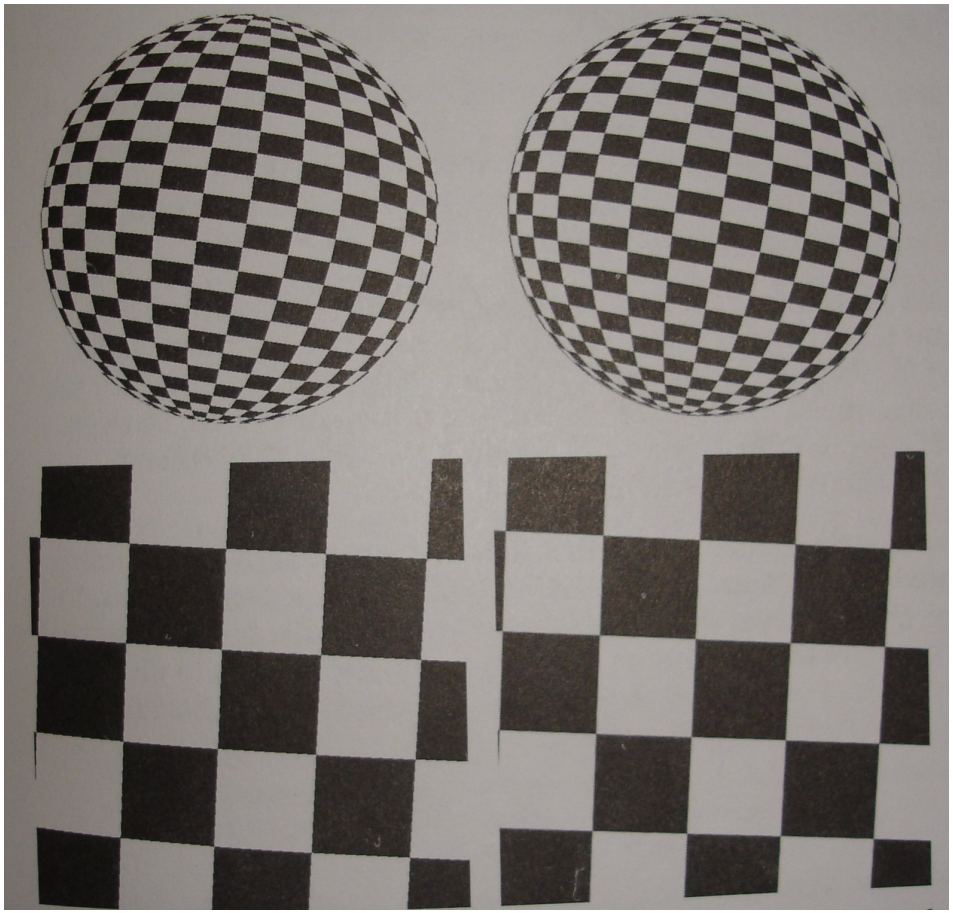
TecGraf PUC RIO

Antialiasing: textura procedural

- Limitação da freqüência (*frequency clamping)*
  - ⇨ Integrais analíticas desconhecidas
  - ⇨ Aplicação passa valor médio da função
    - ✧ Convenientes para funções com médias conhecidas
      - – sine, noise, etc.

Antialiasing: textura procedural

- Limitação da freqüência (*frequency clamping)*
  - ⇨ Exemplo: tabuleiro de xadrez



TecGraf PUC RIO

Antialiasing: textura procedural

- Exemplo: tabuleiro de xadrez

```
uniform vec3  Color1, Color2, AvgColor;
uniform float freq;
varying vec2  TexCoord;

void main (void) {
   vec3 color;
   vec2 fw = fwdith(TexCoord);
   vec2 fuzz = fw * freq * 2.0;          // amount of fuzziness
   float fuzzmax = max(fuzz.x, fuzz,y);
   if (fuzzmax < 0.5)
      ...  // compute pattern color with smooth step
   else
      ... // use average color
}
```

Obs: "if-else" causa aliasing. Temos que suavizar transição.

TecGraf PUC RIO



Noise



Noise

- Introduzido por Ken Perlin
  - Início dos anos 80
- Objetivo
  - Agregar aparência real a objetos
    - Simular imperfeições
    - Simular desgaste com o tempo

Noise

- Requisitos para agregar realismo
  - ⇨ Aparência randômica
    - ✧ Para simular aleatoriedade do fenômeno representado
  - ⇨ Reproduzível
    - ✧ Objetos não podem trocar de aparência se vistos de outros ângulos

Noise

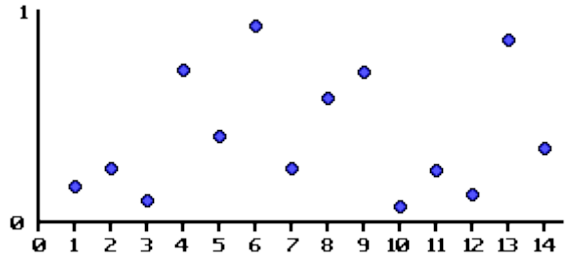
- Para que tem sido usado?
  - Renderização de fenômenos naturais
    - Nuvens, fogo, fumaça
  - Renderização de materiais naturais
    - Mármore, granito, madeira, montanhas
  - Renderização de materiais manufaturados
    - Asfalto, cimento
  - Agregação de imperfeições em padrões perfeitos
    - Agitação, rugosidade, variação de cores
  - Agregação de imperfeições a movimentos
    - Oscilação, variação

Noise

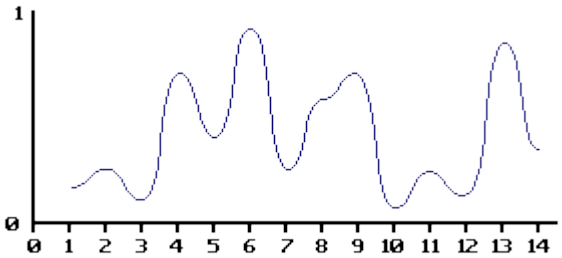
- Propriedades ideiais das funções de noise
  - Função contínua
    - Com aparência de aleatoriedade
    - Não pode reproduzir padrões facilmente indentificados
  - Reproduzível
    - Gerar os mesmo valores para os mesmo parâmetros
  - Saída bem definida, num intervalo fechado
    - Usualmente: [0,1] ou [-1,1]
  - Função isotrópica
    - Invariante por rotação

Noise

- Função de noise
  - ⇨ Número aleatório entre [0,1] para cada valor de x



- Função contínua: interpolação
  - ⇨ Linear: mostraria artefatos
  - ⇨ Cúbica: transições suaves

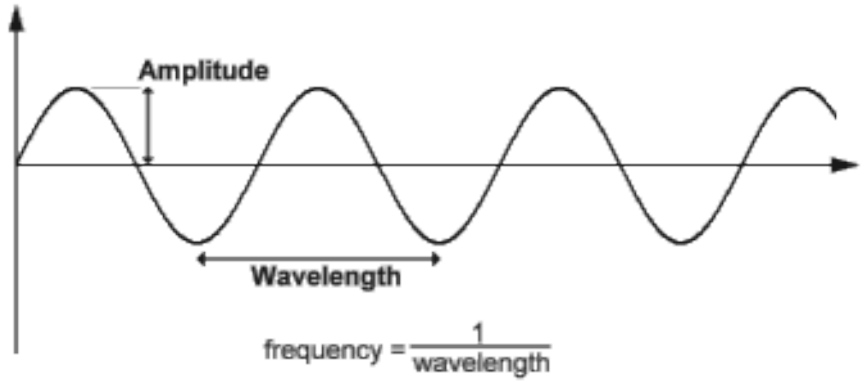


Imagens extraídas de:
http://freespace.virgin.net/hugo.elias/models/m_perlin.htm

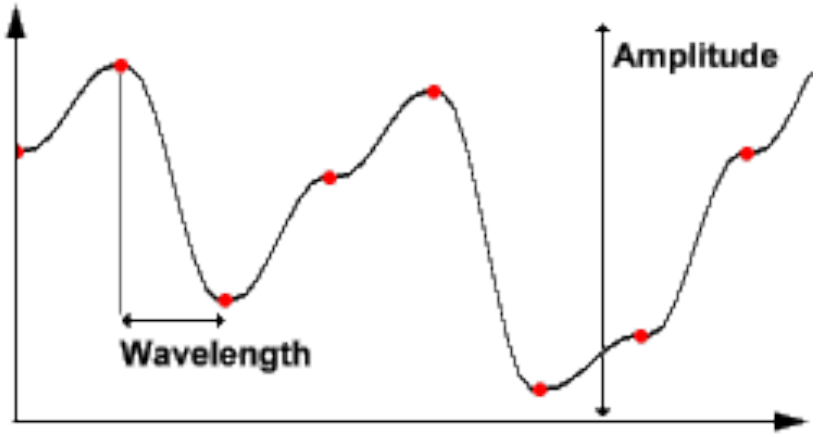
TecGraf PUC RIO

Noise

- Definições: amplitude e freqüência
  - ⇨ Função seno



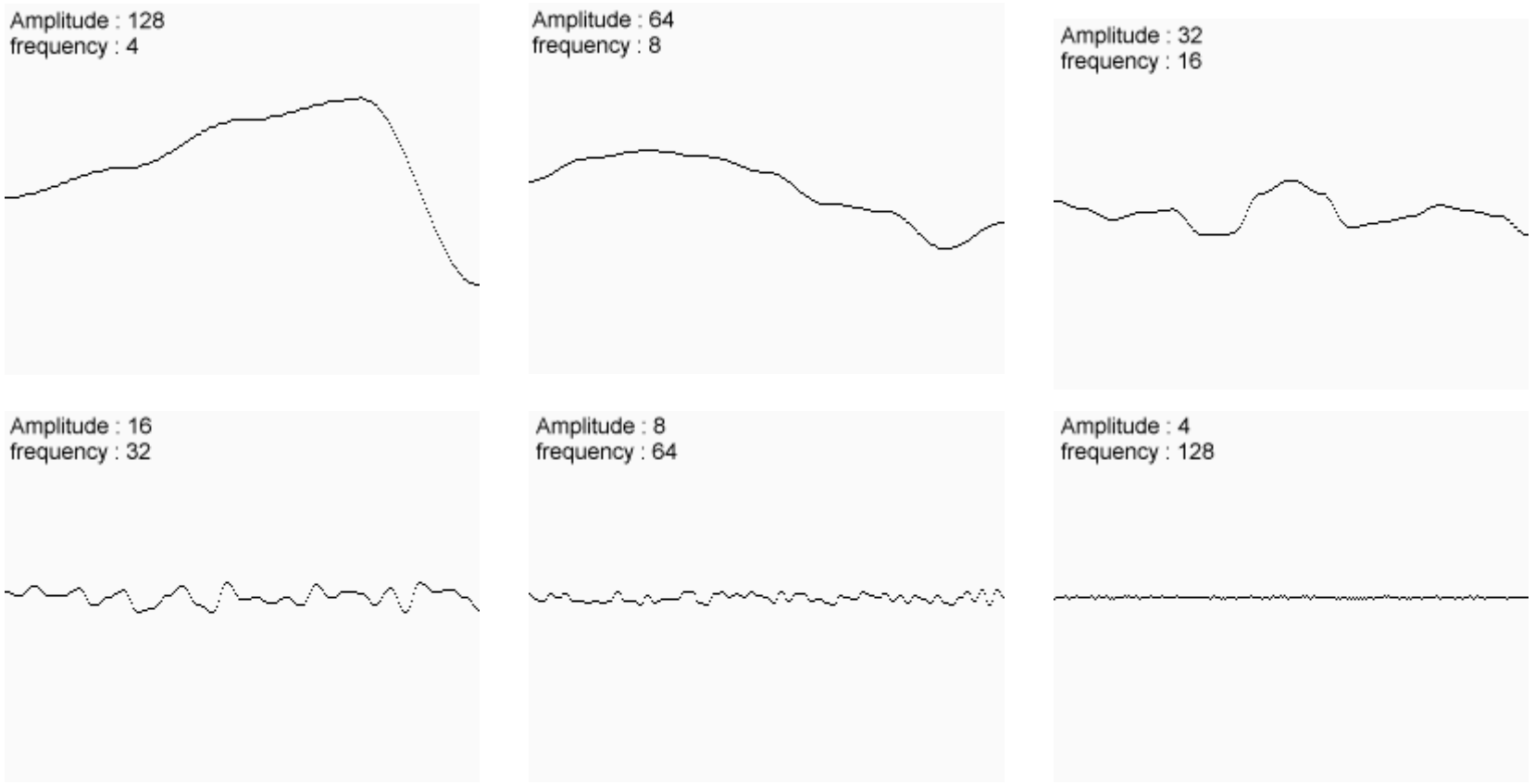
  - ⇨ Função noise



TecGraf PUC RIO

Noise

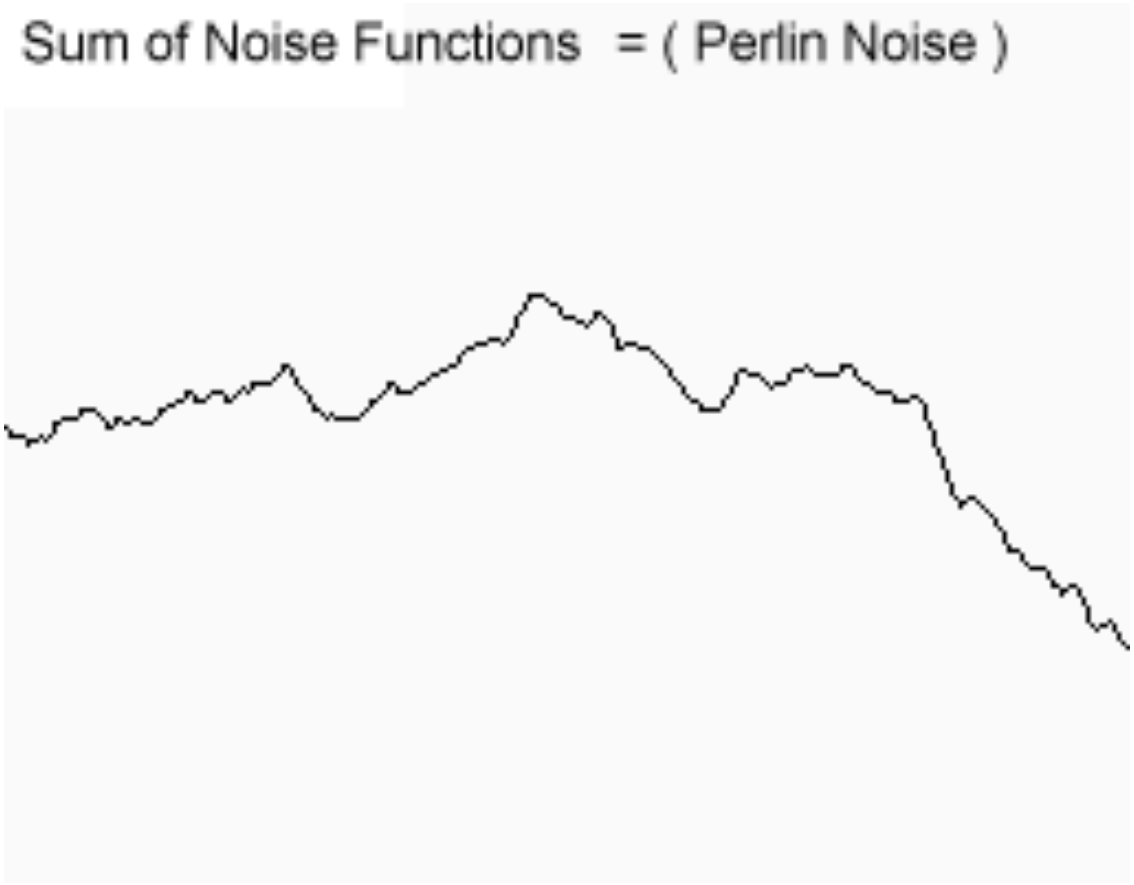
- Variando freqüência e amplitude
  - ⇨ Obtemos um conjunto de funções noise: OCTAVE



TecGraf PUC RIO

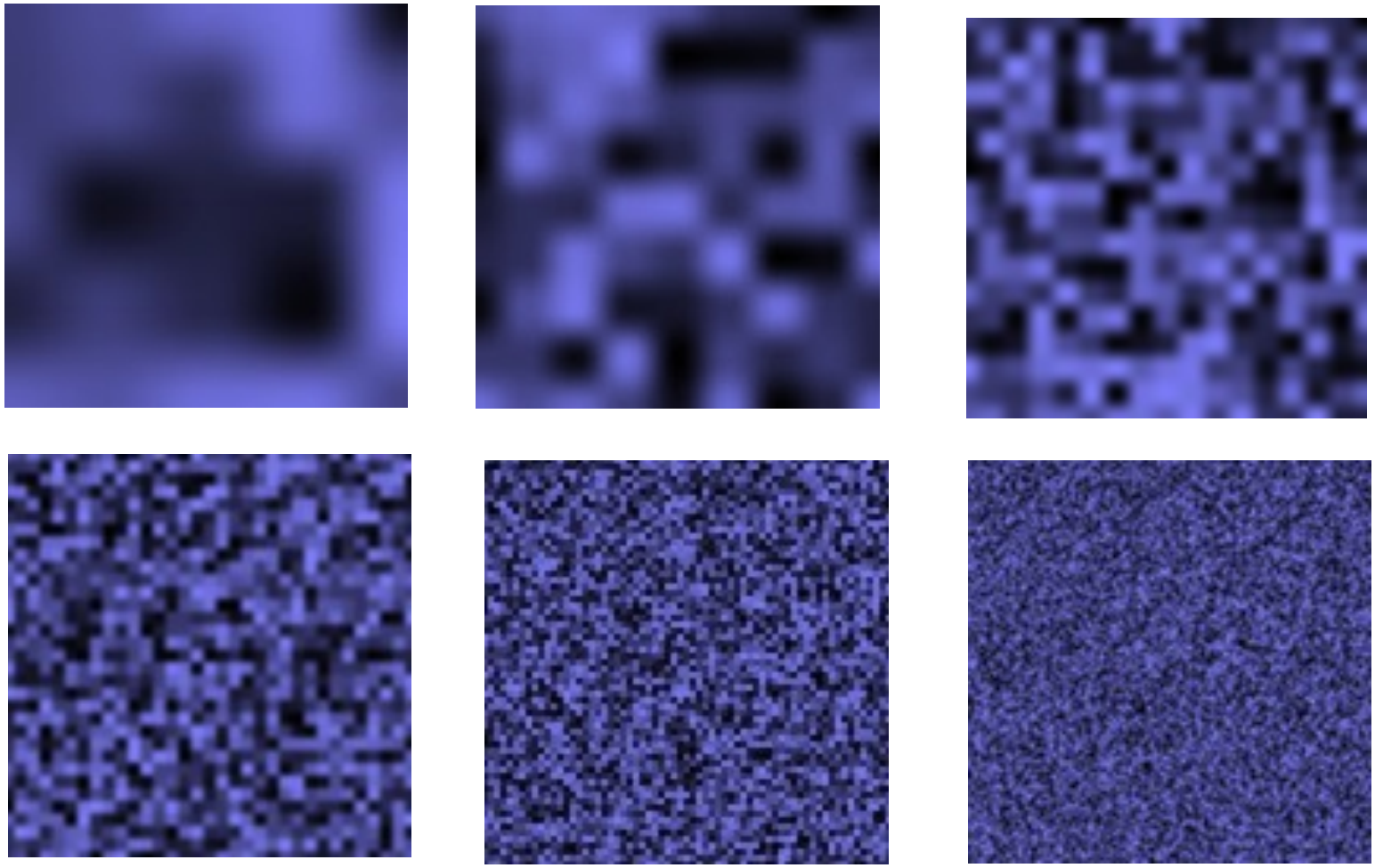
Noise

- Perlin noise
  - ⇨ Somando funções de diferentes freqüências e amplitudes



Noise

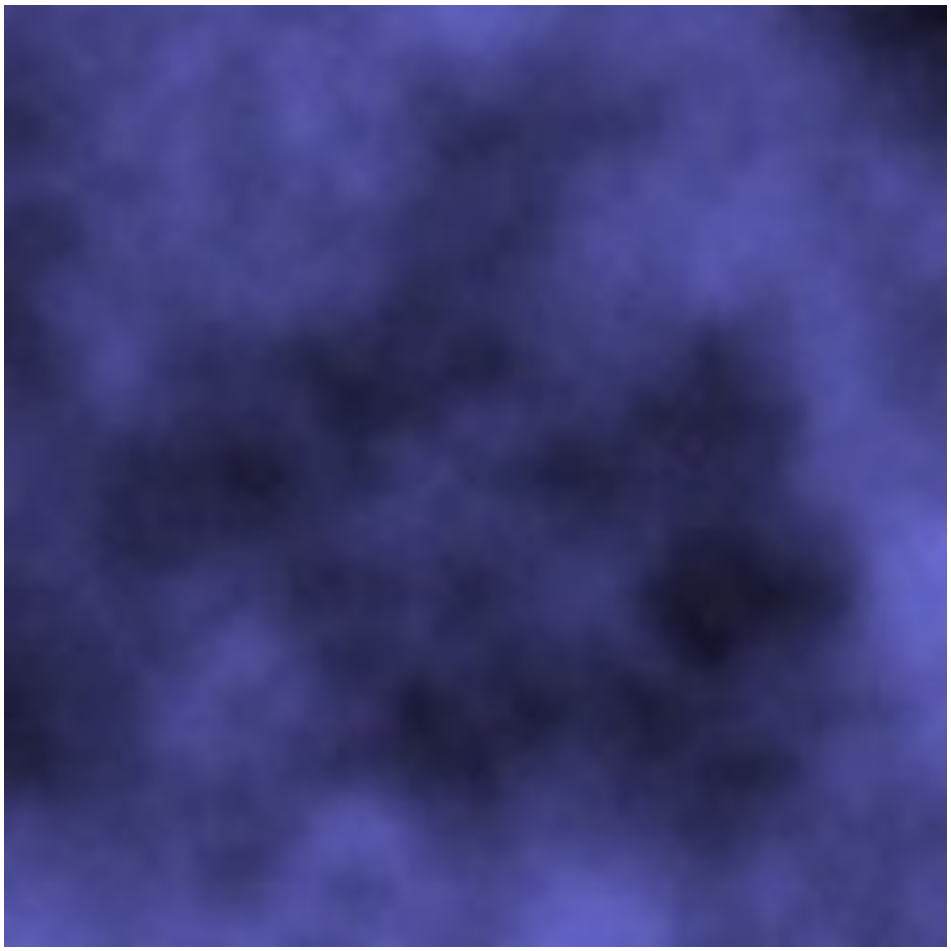
- Funções 2D



TecGraf PUC RIO

Noise

- Perlin noise 2D

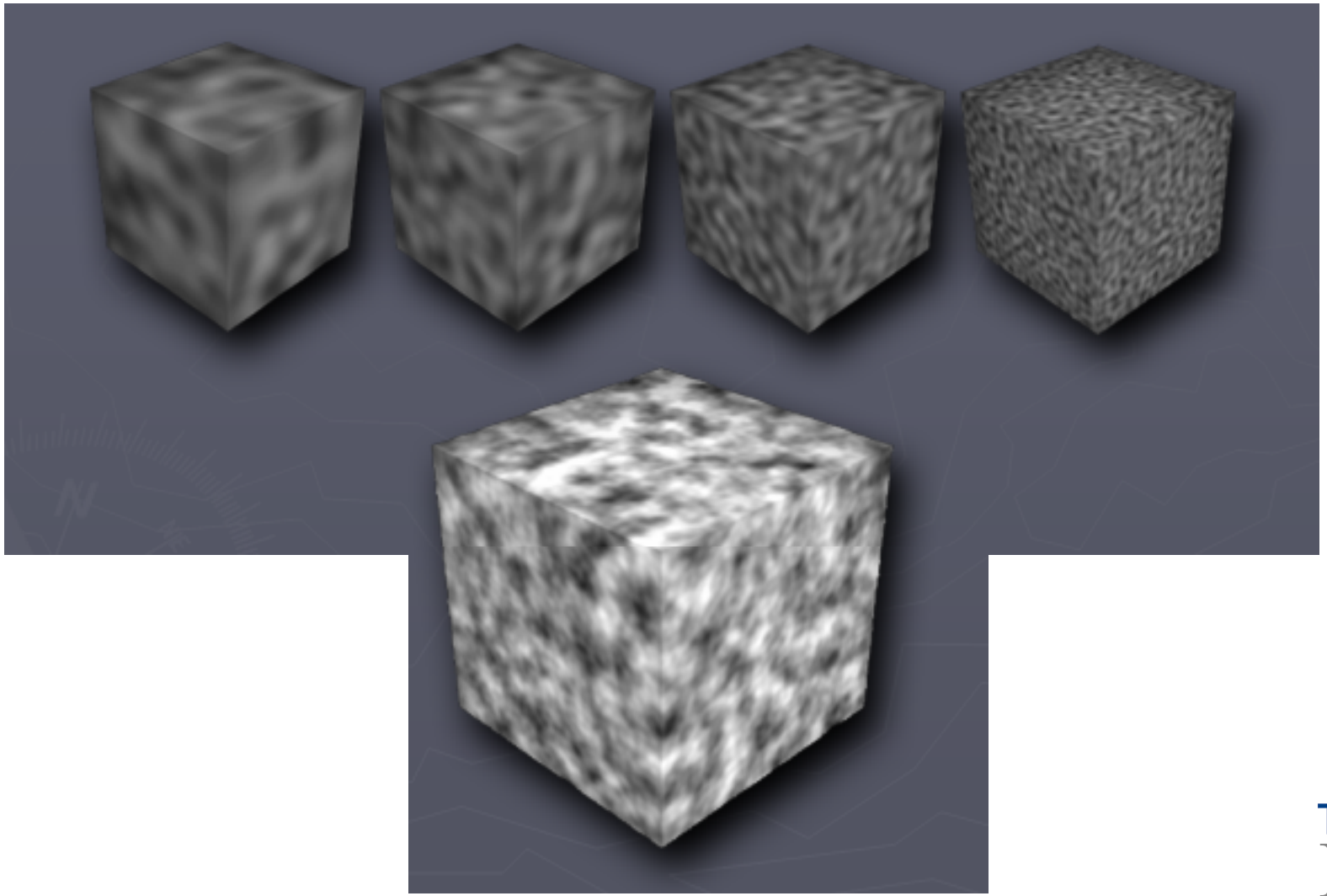


  - ⇨ De forma análoga, podemos gerar outras dimensões
    - ✧ 1D, 2D, 3D, 4D

TecGraf PUC RIO

Noise

- Perlin Noise 3D



TecGraf PUC RIO

Noise

- Perlin noise functions

```
function Noise1D(integer x)
        x = (x<<13) ^ x;
        return ( 1.0 - ( (x * (x * x * 15731 + 789221) + 1376312589) & 7fffffff) / 1073741824.0);
end function
```

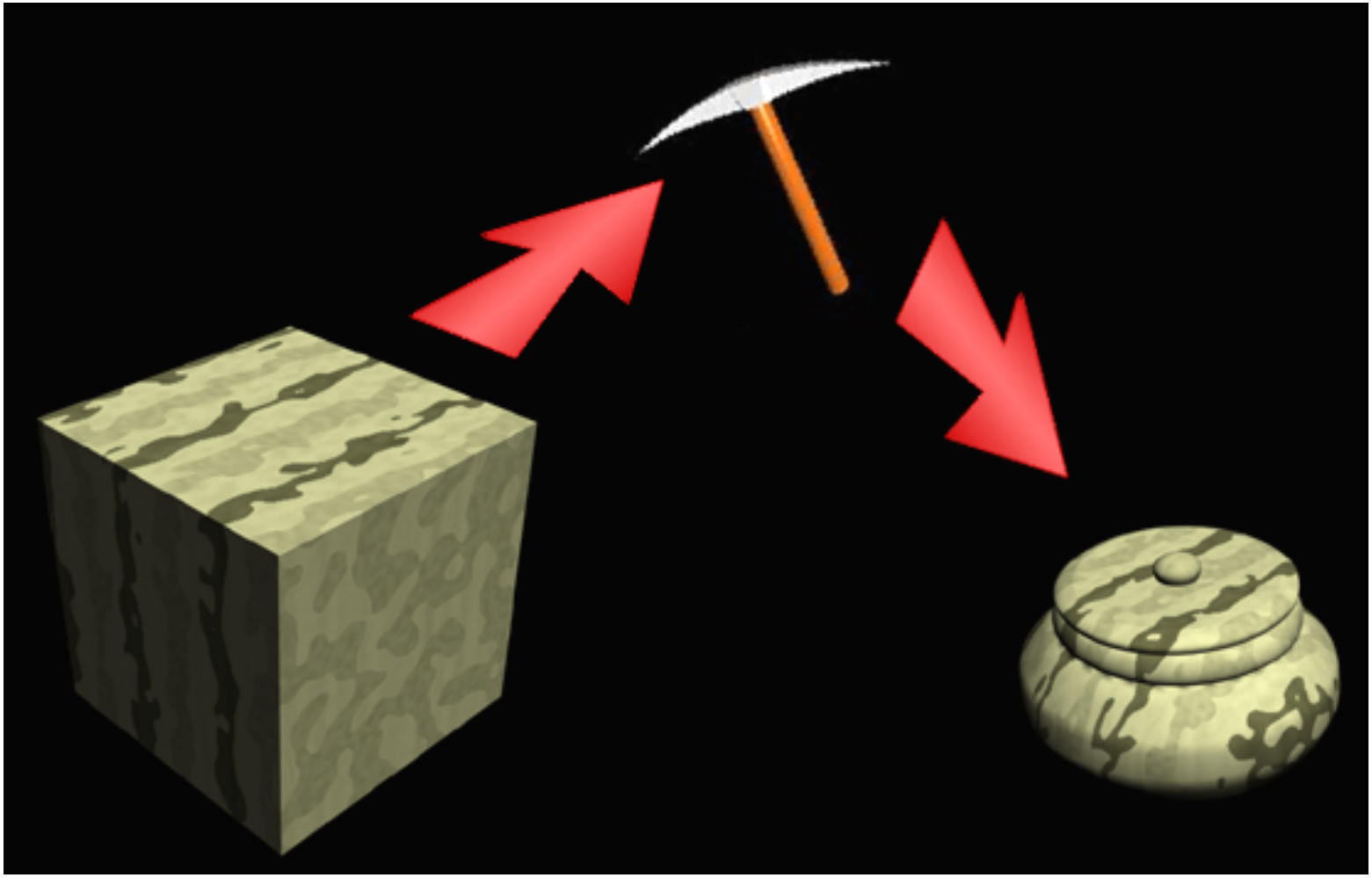
```
function Noise2D(integer x, integer y)
        n = x + y * 57
        n = (n<<13) ^ n;
        return ( 1.0 - ( (n * (n * n * 15731 + 789221) + 1376312589) & 7fffffff) / 1073741824.0);
end function
```

TecGraf PUC RIO

Noise em shaders

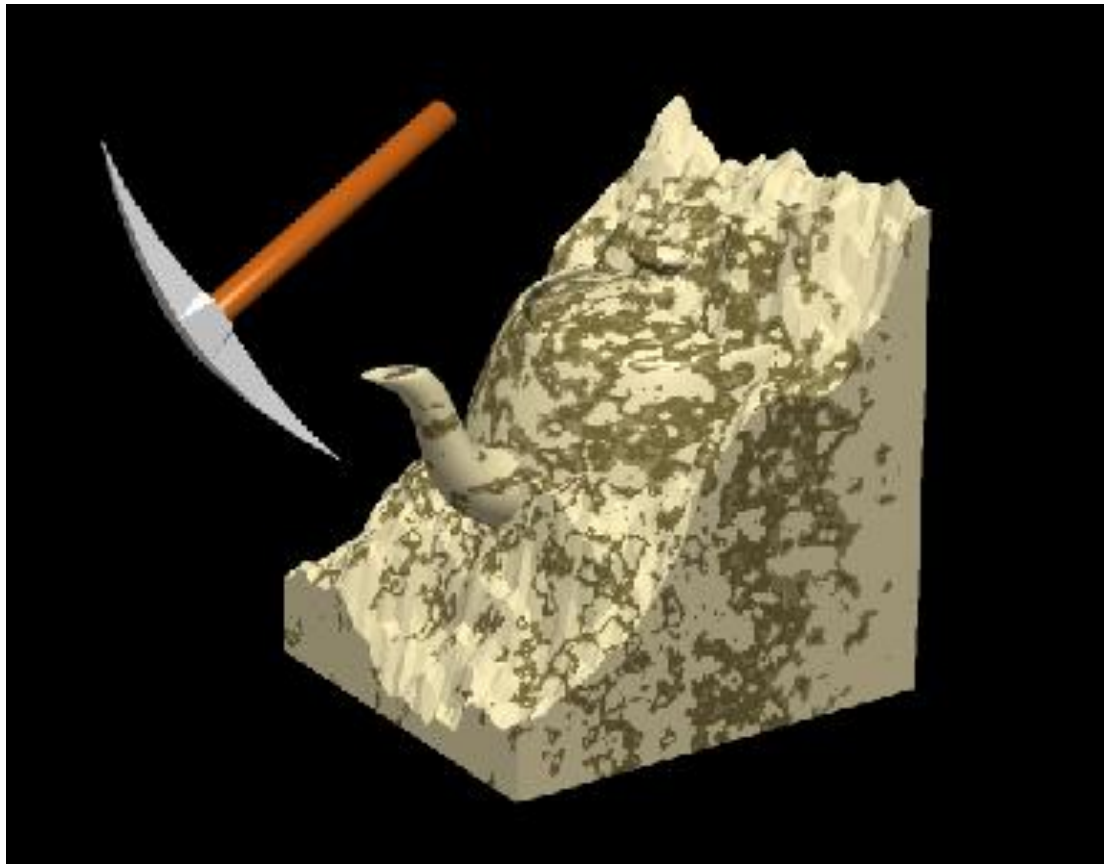
- Estratégias
  - Usar funções built-in
    - Já disponível?
  - Escrever nossas funções de noise
    - Viável com a programabilidade atual?
    - Ineficiente
  - Codificar função noise em texturas
    - Problemas com baixa taxa de amostragem
    - Garante mesma aparência entre GPU diferentes
    - Tende a apresentar melhor desempenho

Aplicações



TecGraf PUC RIO

Aplicações



TecGraf PUC RIO

Noise: exemplos

- Função noise codificada numa textura 3D
  - Componentes RGBA
    - R: 1st octave: f = 4; a = 0.5;
      - Média: 0.25
    - G: 2nd octave: f = 8; a = 0.25;
      - Média: 0.125
    - B: 3rd octave: f = 16; a = 0.125
      - Média: 0.0625
    - A: 4th octave: f = 32; a = 0.0625
      - Média: 0.03125

Noise: exemplos

- Textura de nuvens



TecGraf PUC RIO

Noise: textura de nuvens

- Vertex shader

```
out float LightIntensity;
out vec3  MCposition;

uniform vec3  LightPos;
uniform float Scale;

void main(void)
{
    vec3 ECposition = vec3 (gl_ModelViewMatrix * gl_Vertex);
    MCposition      = vec3 (gl_Vertex) * Scale;
    vec3 tnorm      = normalize(vec3 (gl_NormalMatrix * gl_Normal));
    LightIntensity  = dot(normalize(LightPos - ECposition), tnorm);
    LightIntensity *= 1.5;
    gl_Position     = ftransform();
}
```

TecGraf PUC RIO

Noise: textura de nuvens

- Fragment shader

```
in float LightIntensity;
in vec3  MCposition;
uniform sampler3D Noise;
uniform vec3 SkyColor;       // (0.0, 0.0, 0.8)
uniform vec3 CloudColor;     // (0.8, 0.8, 0.8)

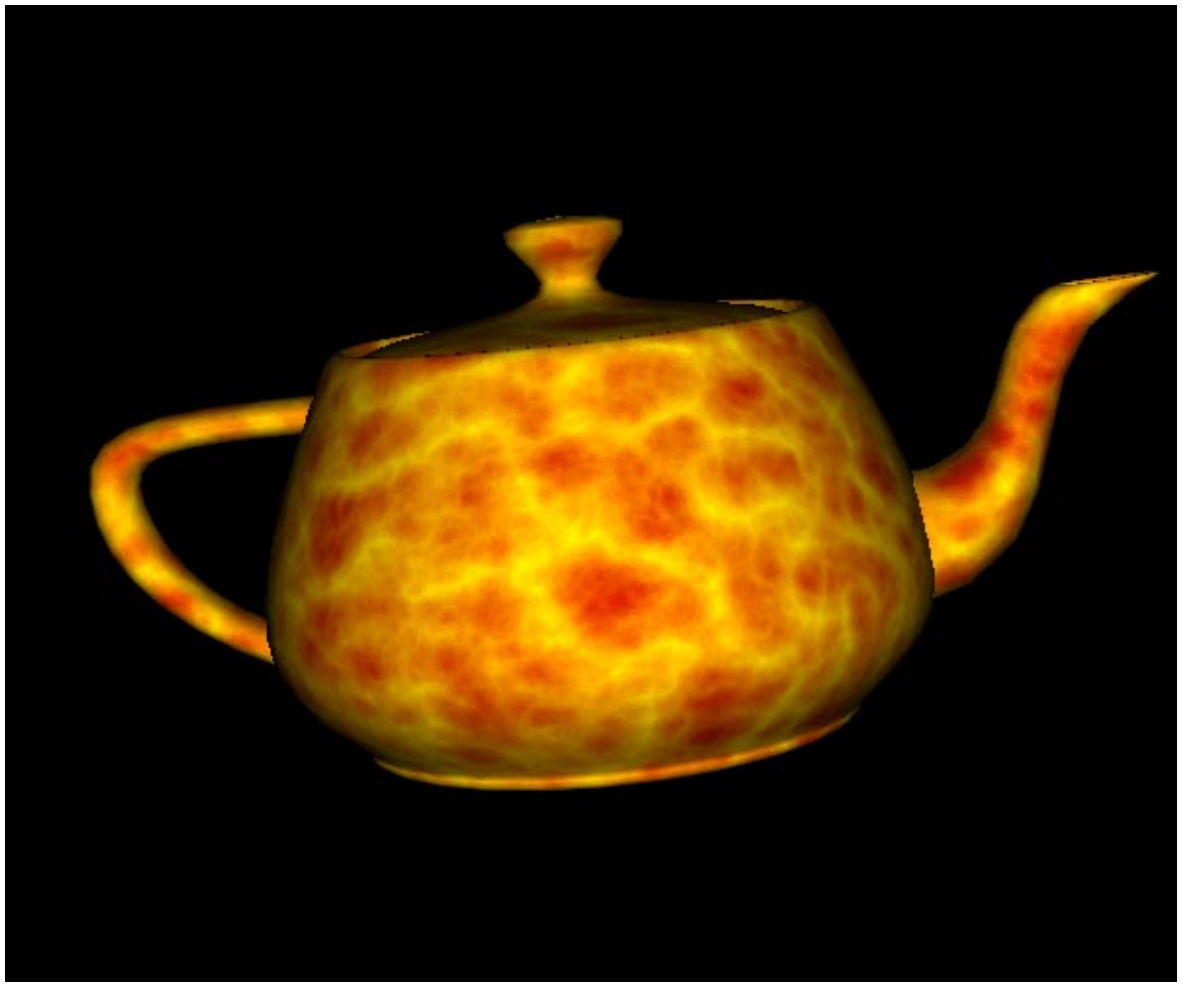
void main (void)
{
    vec4  noisevec  = texture3D(Noise, MCposition);
    float intensity = (noisevec[0] + noisevec[1] +
                       noisevec[2] + noisevec[3] + 0.03125) * 1.5;
    vec3 color   = mix(SkyColor, CloudColor, intensity) * LightIntensity;
    gl_FragColor = vec4 (color, 1.0);
}
```

- ⇨ 0.03125 adicional: média das octantes restantes
- ⇨ 1.5 como fator de escala para acentuar efeito

TecGraf PUC RIO

Noise: turbulência

- Textura da superfície do sol



TecGraf PUC RIO

Noise: turbulência

- Efeito de turbulência
  - ⇨ Usar valor absoluto de cada função
    - ✧ Simula dobras, vincos, rucas
    - ✧ Usada para simular fogos, larvas
  - ⇨ Para cada octave
    - ✧ Translada para ter zero como valor médio
    - ✧ Tira absoluto
      - $v_{new} = abs(v - v_{avg})$

Noise: turbulência

- Fragment shader

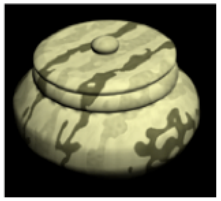
```
in float LightIntensity;
in vec3  MCposition;
uniform sampler3D Noise;
uniform vec3 Color1;        // yellow: (0.8, 0.7, 0.0)
uniform vec3 Color2;        // red: (0.6, 0.1, 0.0)
uniform float NoiseScale;   // 1.2

void main (void)
{
   vec4 noisevec = texture3D(Noise, MCposition * NoiseScale);
   float intensity = abs(noisevec[0] - 0.25) + abs(noisevec[1] - 0.125) +
                     abs(noisevec[2] - 0.0625) + abs(noisevec[3] - 0.03125);
   intensity    = clamp(intensity * 6.0, 0.0, 1.0);
   vec3 color   = mix(Color1, Color2, intensity) * LightIntensity;

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

TecGraf PUC RIO

Noise: aparência de mármore



- Noise adicionada da função periódica seno
  - ⇨ “Noisy oscillating function”
  - ⇨ Simula a coloração em “veias” de mármore

```
    ...
    vec4 noisevec = texture3D(Noise, MCposition * NoiseScale);
    float intensity = abs(noisevec[0] - 0.25) + abs(noisevec[1] - 0.125) +
                      abs(noisevec[2] - 0.0625) + abs(noisevec[3] - 0.03125);
    float sineval  = sin(MCposition * 6.0 + intensity * 12) * 0.5 + 0.5;
    vec3 color   = mix(veinColor, marbleColor, sineval) * LightIntensity;

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

TecGraf PUC RIO

Noise: granito

- Superfície cinzenta de rocha natural



TecGraf PUC RIO

Noise: granito

- Fragment shader
  - ⇨ Usa apenas a “octave” de maior freqüência
    - ✧ Escala valor para obter intensidade apropriada

```
in float LightIntensity;
in vec3  MCposition;

uniform sampler3D Noise;
uniform float NoiseScale;

void main(void)
{
    vec4  noisevec  = texture3D(Noise, NoiseScale * MCposition);
    float intensity = min(1.0, noisevec[3] * 18.0);
    vec3  color     = vec3 (intensity * LightIntensity);

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

TecGraf PUC RIO

Noise: madeira

- Textura para objetos manufaturados em madeira



TecGraf PUC RIO

Noise: madeira

- Idéia básica
  - Textura composta de cilindros concêntricos
    - Áreas claras e áreas escuras
  - Noise adicionado para obter variação natural
  - Assume-se centro da “árvore” no eixo y
  - Alta freqüência simulando serragem

Noise: madeira

- Fragment shader

```
in float LightIntensity;
in vec3  MCposition;

uniform sampler3D Noise;

uniform vec3  LightWoodColor;     // 0.6, 0.3, 0.1
uniform vec3  DarkWoodColor;      // 0.4, 0.2, 0.07
uniform float  RingFreq;          // 4.0
uniform float  LightGrains;       // 1.0
uniform float  DarkGrains;        // 0.0
uniform float  GrainThreshold;    // 0.5
uniform vec3  NoiseScale;         // 0.5, 0.1, 0.1
uniform float  Noisiness;         // 3.0
uniform float  GrainScale;        // 27.0
```

Noise: madeira

- Fragment shader

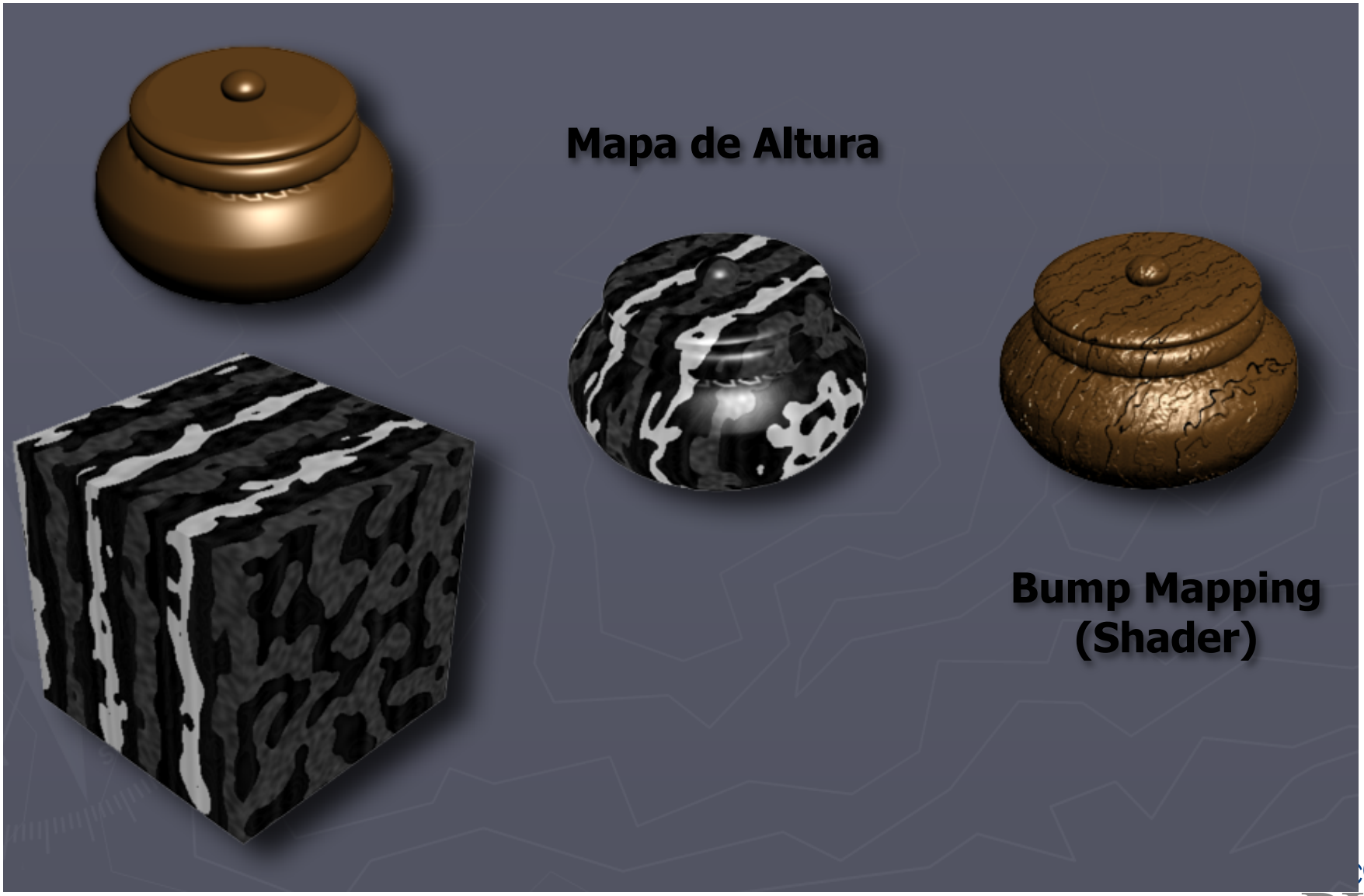
```
void main(void)
{
    vec3 noisevec = vec3 (texture3D(Noise, MCposition * NoiseScale) * Noisiness);
    vec3 location = MCposition + noisevec;
    float dist = sqrt(location.x * location.x + location.z * location.z);
    dist *= RingFreq;
    float r = fract(dist + noisevec[0] + noisevec[1] + noisevec[2]) * 2.0;
    if (r > 1.0) r = 2.0 - r;
    vec3 color = mix(LightWoodColor, DarkWoodColor, r);
    // efeito de serragem (alta freqüência)
    r = fract((MCposition.x + MCposition.z) * GrainScale + 0.5);
    noisevec[2] *= r;
    if (r < GrainThreshold)   color += LightWoodColor * LightGrains * noisevec[2];
    else                      color -= LightWoodColor * DarkGrains * noisevec[2];
    color *= LightIntensity;
    gl_FragColor = vec4 (color, 1.0);
}
```



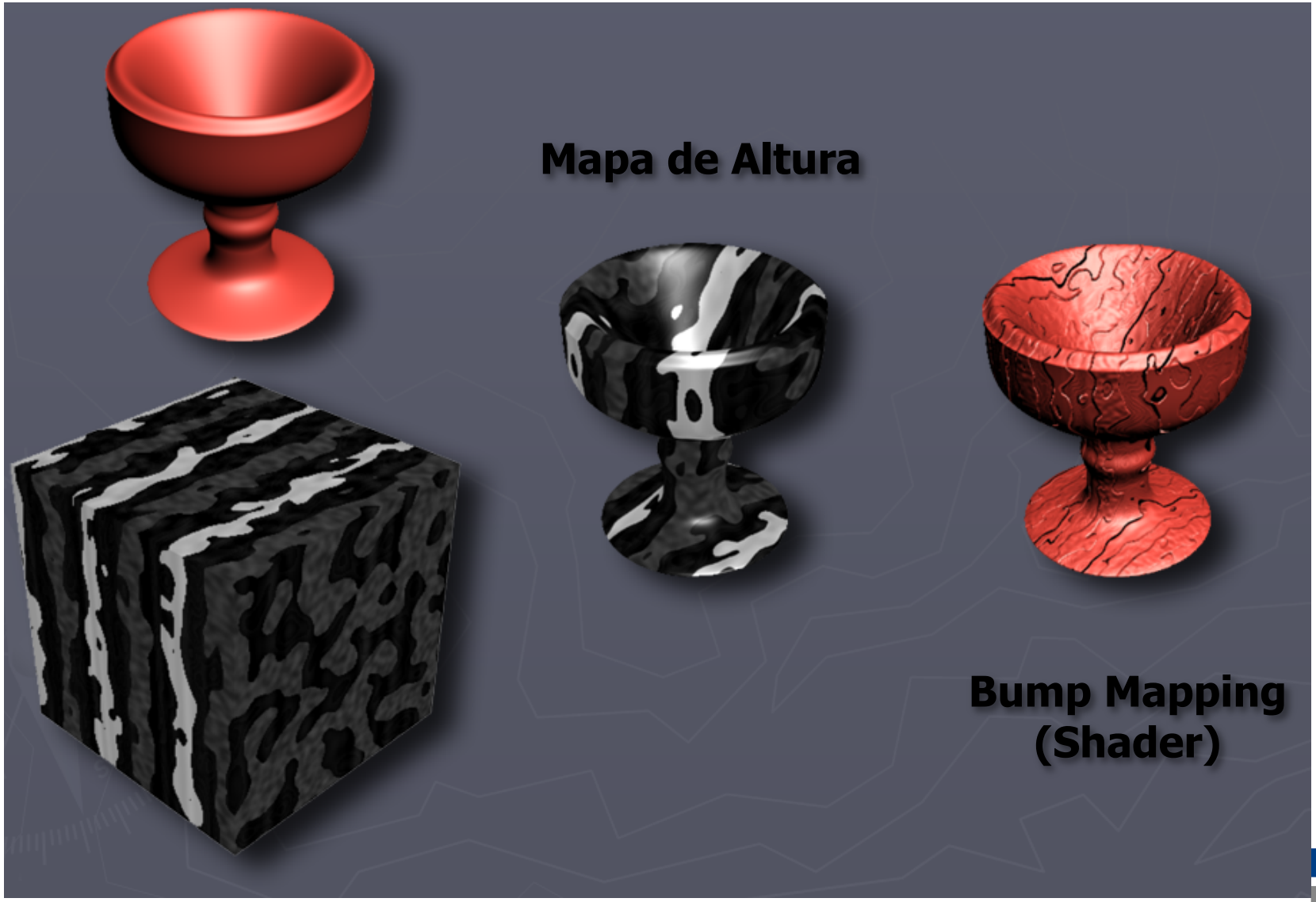
Bump Procedural

Marcio Pereira de Araujo
Waldemar Celes

Idéia

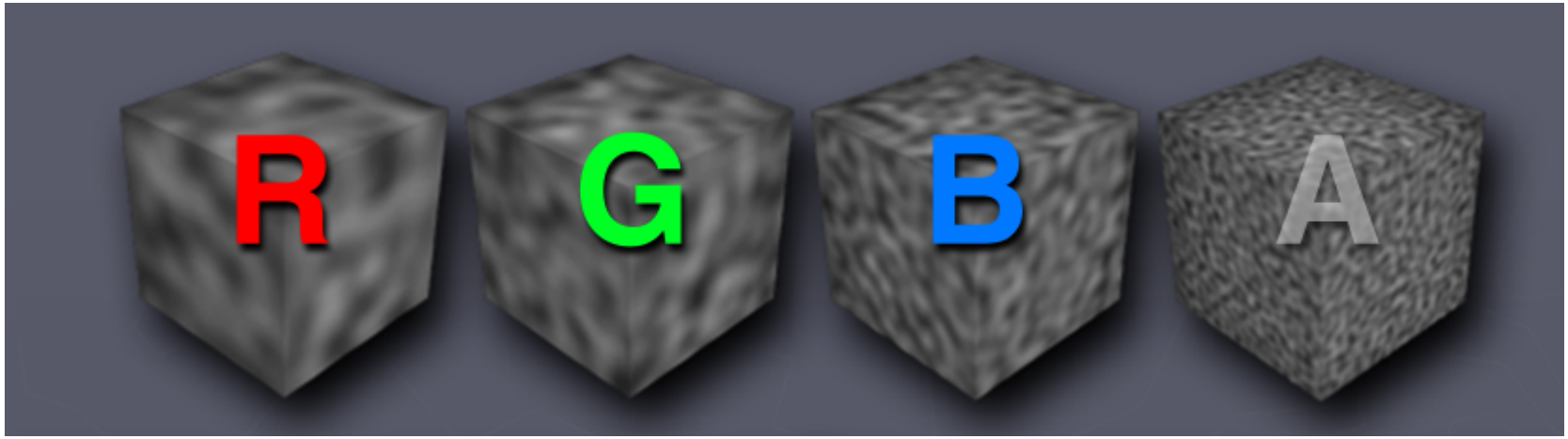


Idéia



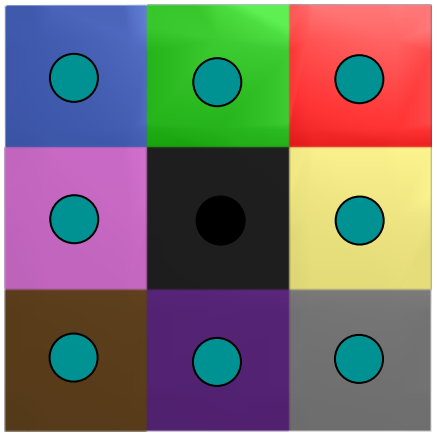
Solução Atual

- Uma Textura 3D contendo 4 Octaves
  - Idéia
    - Mapear as 4 Octaves nas componentes RGBA de uma Textura 3D



Solução Atual

- Bump Mapping de um Mapa de Altura
  - ⇨ Objeto 3D mapeado por um “Mapa de Altura”
  - ⇨ Descobrir a normal do pixel a partir do “Mapa de Altura”
    - ✧ Consultar na superfície do objeto os valores dos texels vizinhos



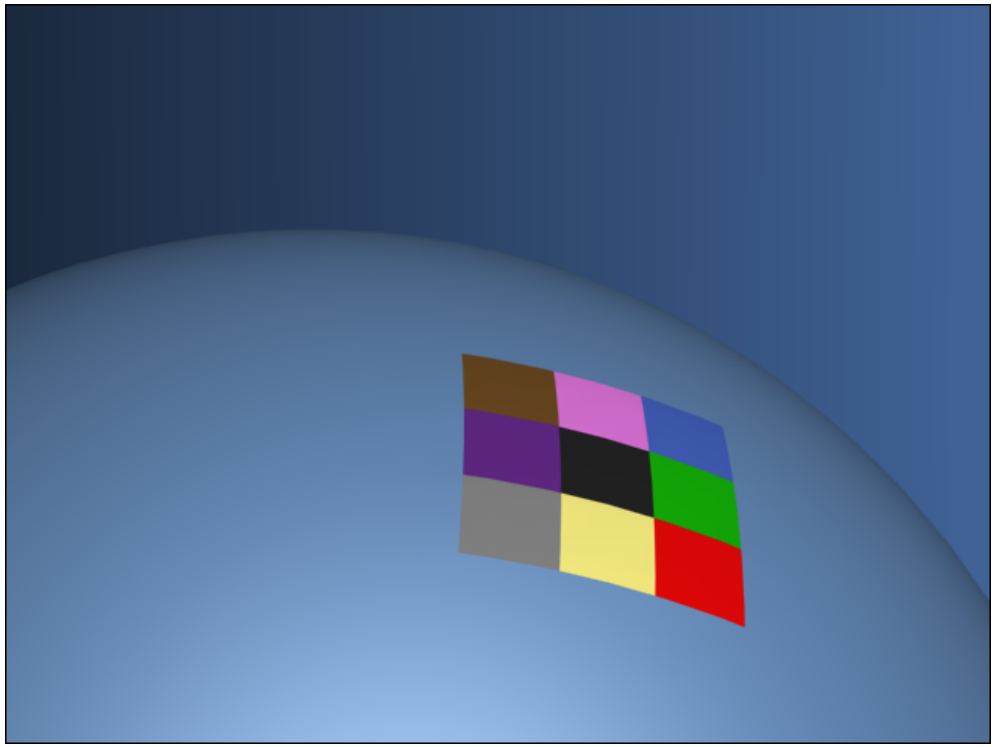
Texel Corrente

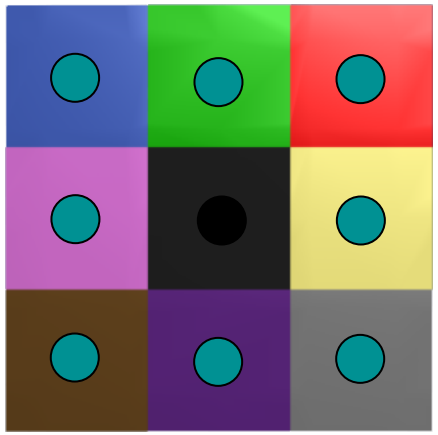
Texels Vizinhos

TecGraf PUC RIO

Solução Atual

- Bump Mapping de um Mapa de Altura
  - ⇨ Consultar na superfície do objeto os valores dos texels vizinhos





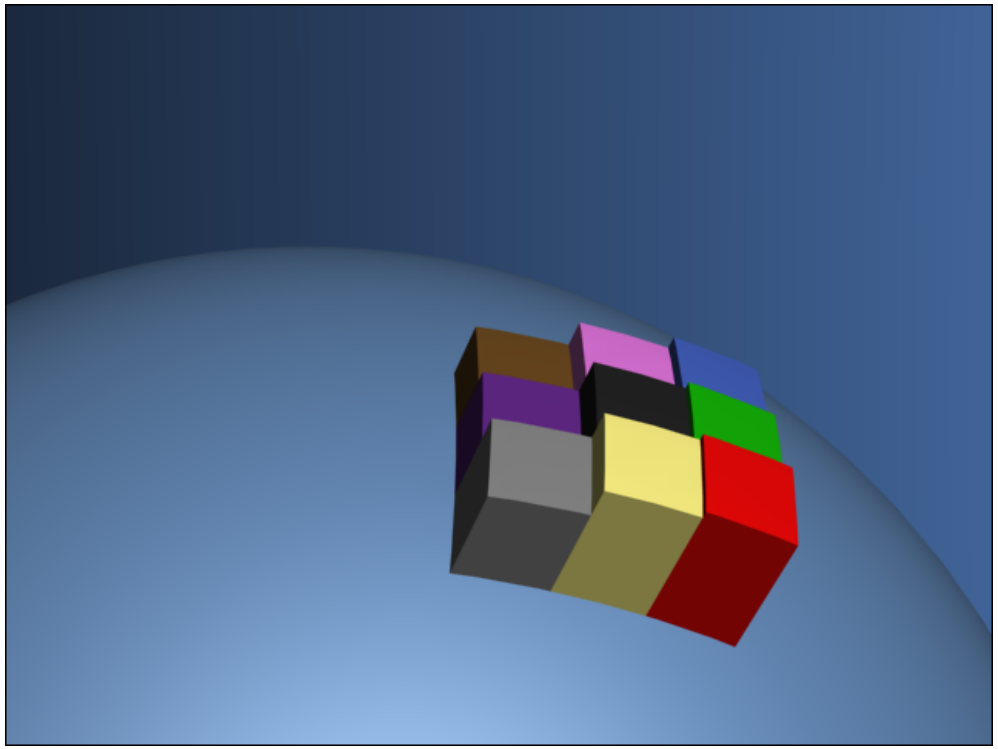
● **Texel Corrente**

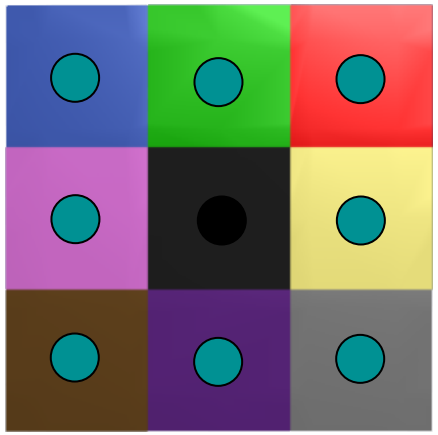
● **Texels Vizinhos**

TecGraf PUC RIO

Solução Atual

- Bump Mapping de um Mapa de Altura
  - ⇨ Consultar na superfície do objeto os valores dos texels vizinhos



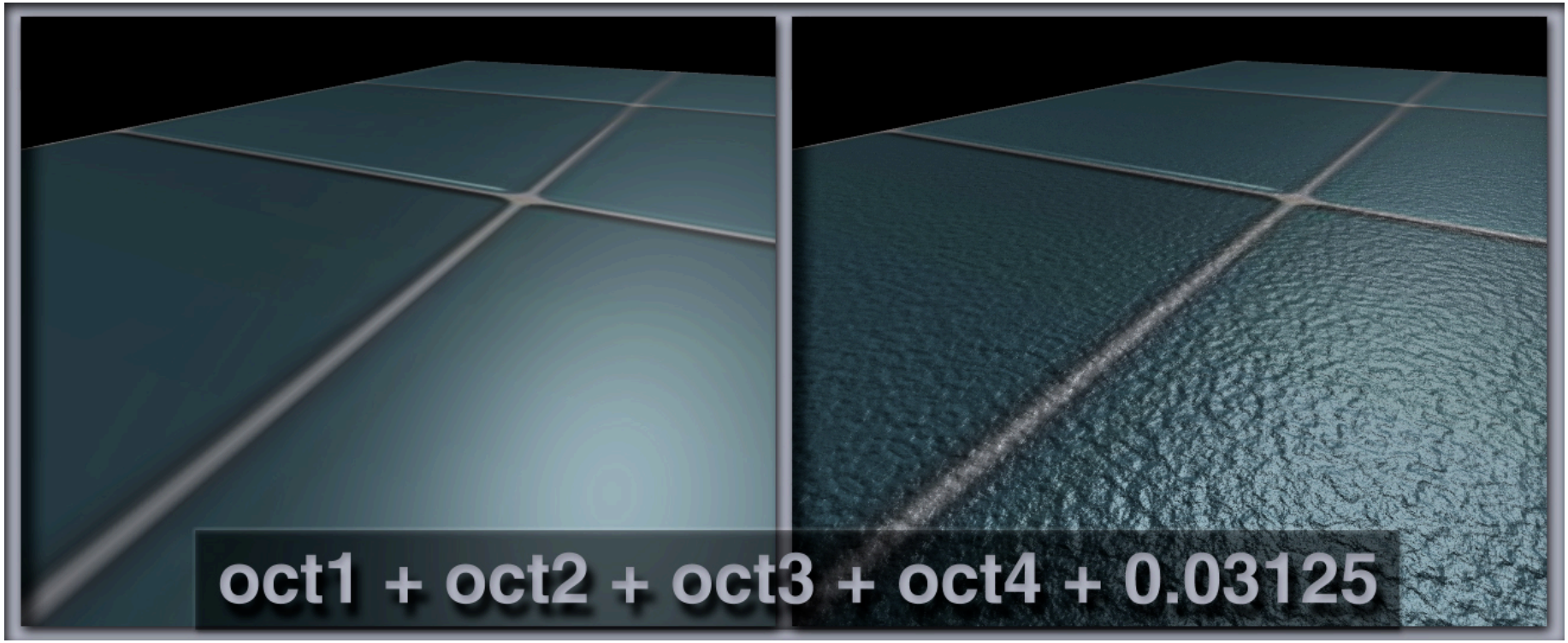


● **Texel Corrente**

● **Texels Vizinhos**

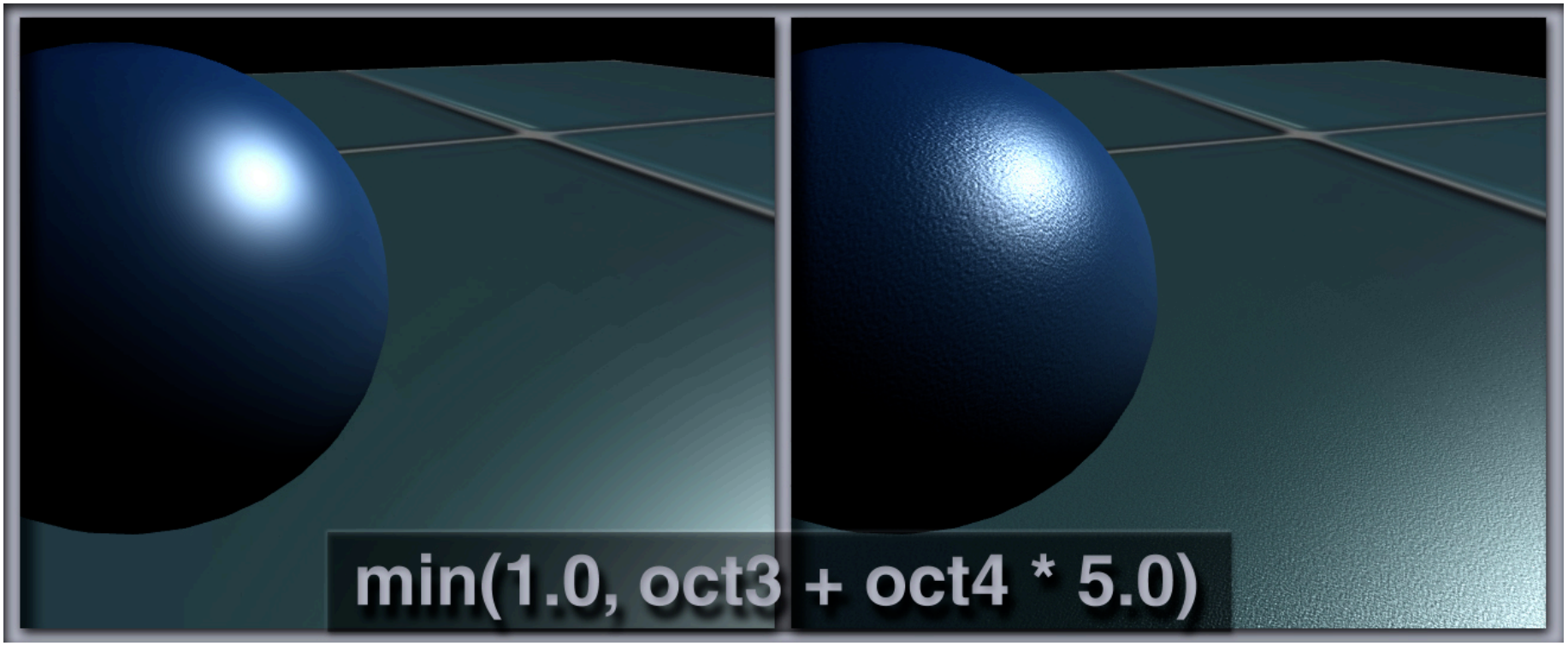
TecGraf PUC RIO

Exemplos

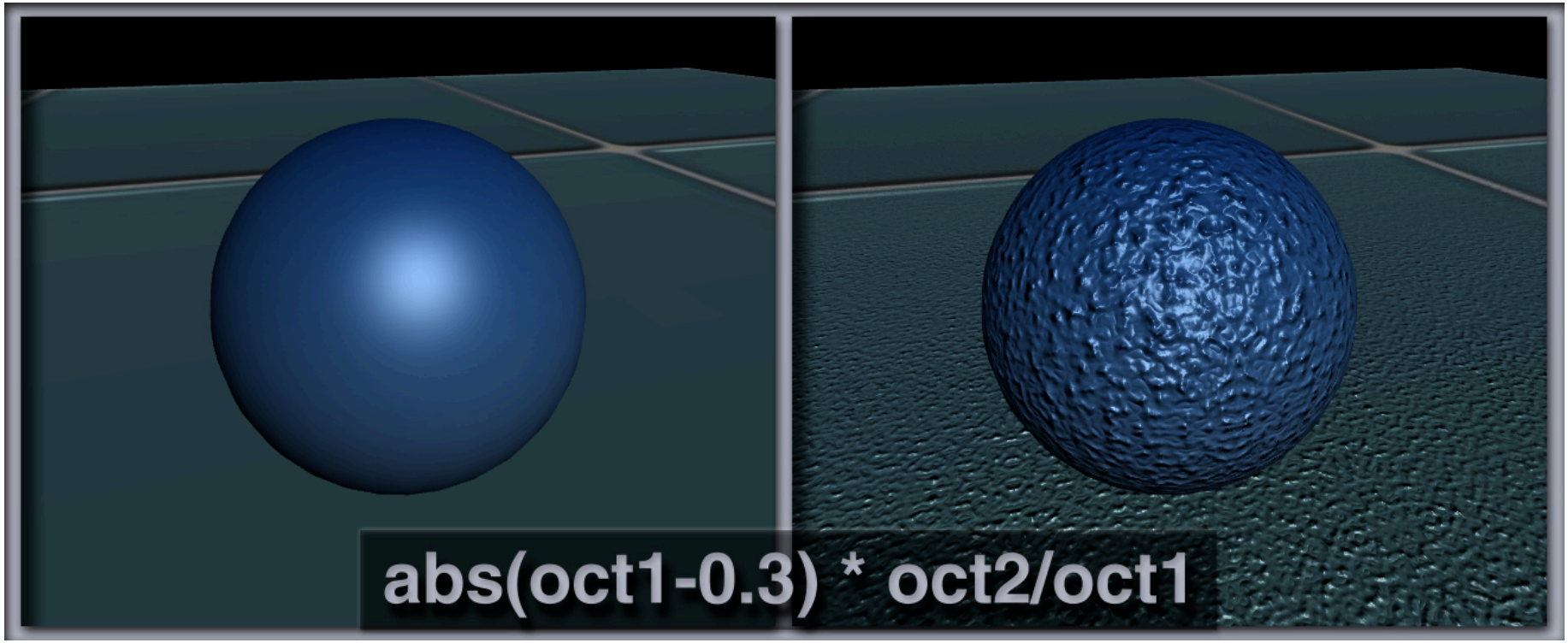


TecGraf
PUC
RIO

Exemplos

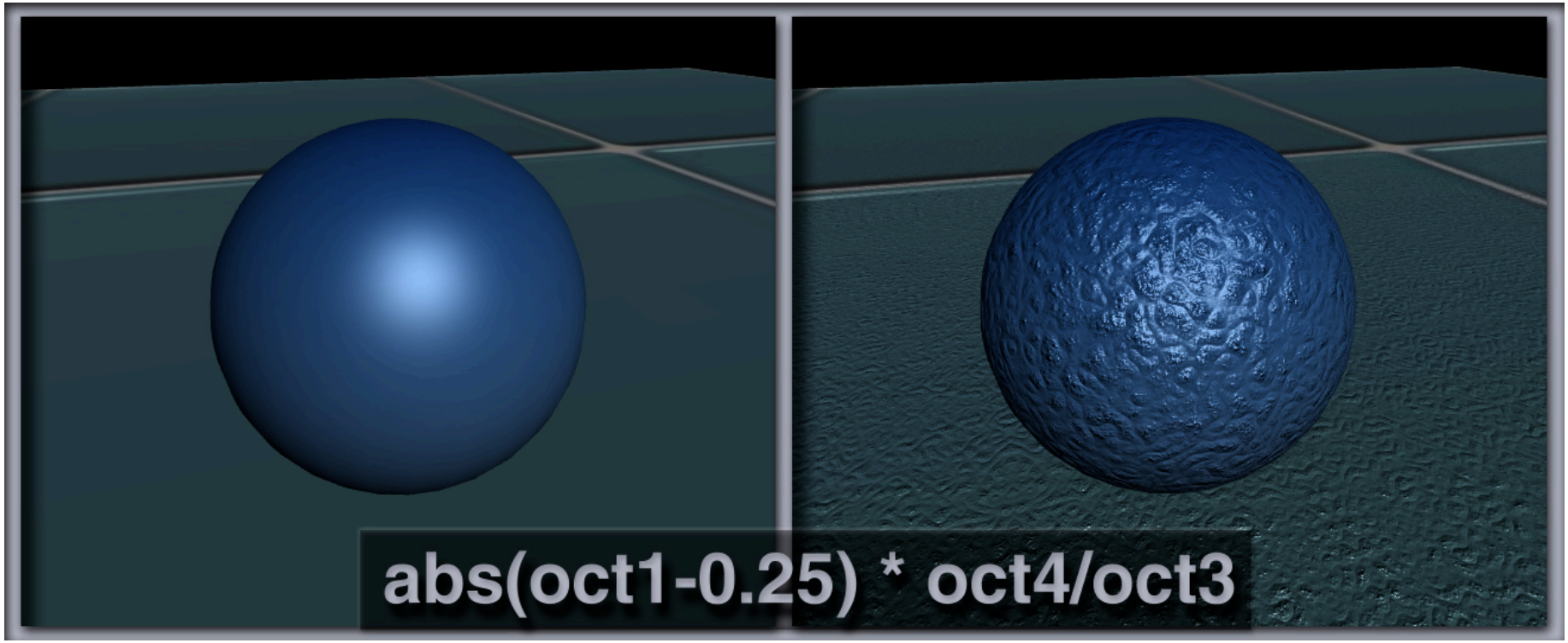


Exemplos



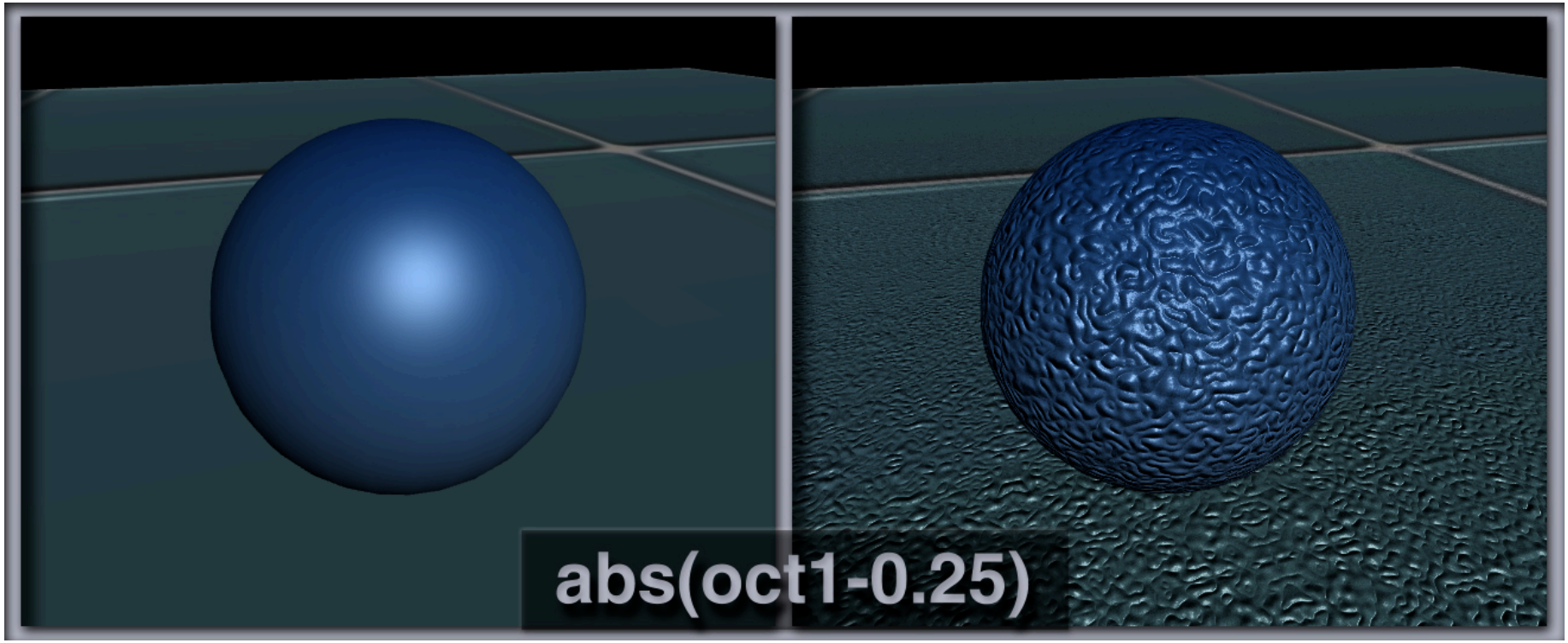
TecGraf
PUC
RIO

Exemplos



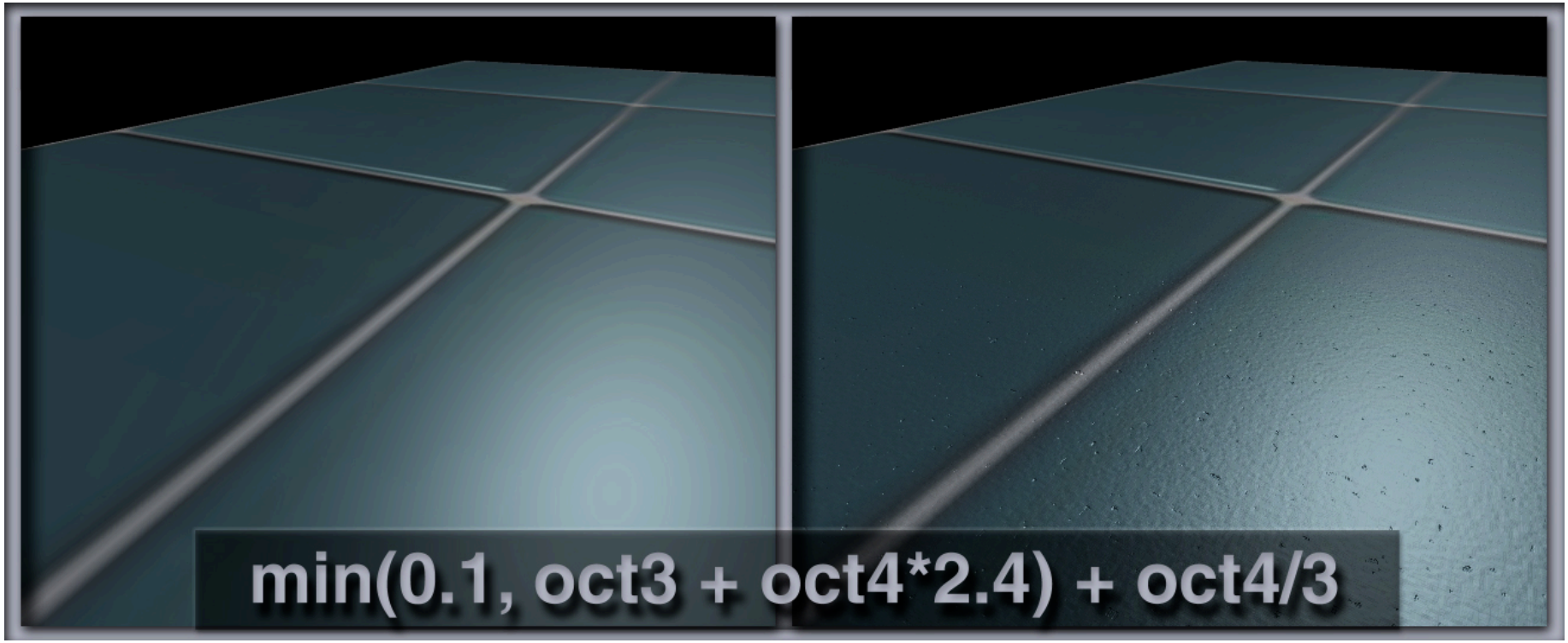
TecGraf
PUC
RIO

Exemplos



TecGraf
PUC
RIO

Exemplos



TecGraf PUC RIO