



---

# Procedural Texturing and Shading

---

Johns Hopkins Department of Computer Science  
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



---

# Procedural Texturing/Shading

**Paradigm for programmability in the graphics pipeline**

**Allows for a wide variety of surface materials and embellishments**

**May be facilitated by a custom shading language**

- e.g. Pixar's RenderMan

---

Johns Hopkins Department of Computer Science  
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



## **Potential Advantages of Procedural Textures**

**Compact representation**

**No fixed resolution**

**No fixed area**

**Parameterized - generates class of related  
textures**

---

Johns Hopkins Department of Computer Science  
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



## **Disadvantages of Procedural Textures**

**Difficult to build and debug**

**Surprising results**

**Slow evaluation**

**Antialiasing handled manually**

---

Johns Hopkins Department of Computer Science  
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



## Procedural Texture Conventions

---

### **Avoid conditionals**

- **Convert to mathematical functions when possible**
- **Makes anti-aliasing easier**

### **Parameterize rather than building in constants**

- **Assign reasonable defaults which may be overridden**

---

Johns Hopkins Department of Computer Science  
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



## Simple Building Blocks

---

**Mix (lerp)**

**Step, smoothstep, pulse**

**Min, max, clamp, abs**

**Sin, cos**

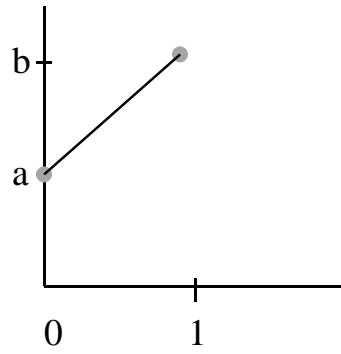
**Mod, floor, ceil**

---

Johns Hopkins Department of Computer Science  
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



## Mix

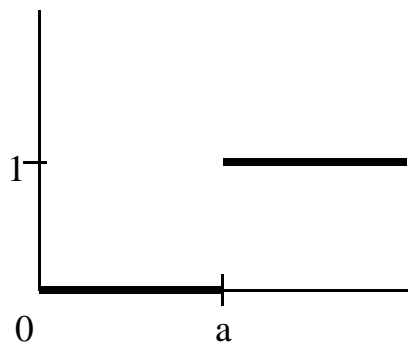


$\text{mix}(a,b,x)$

Johns Hopkins Department of Computer Science  
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



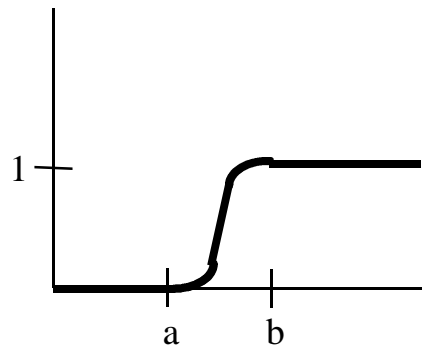
## Step



Johns Hopkins Department of Computer Science  
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



## Smoothstep

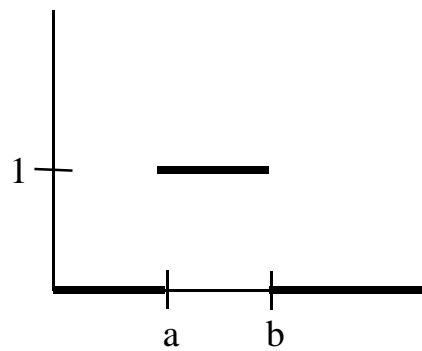


$\text{smoothstep}(a,b,x)$

Johns Hopkins Department of Computer Science  
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



## Pulse

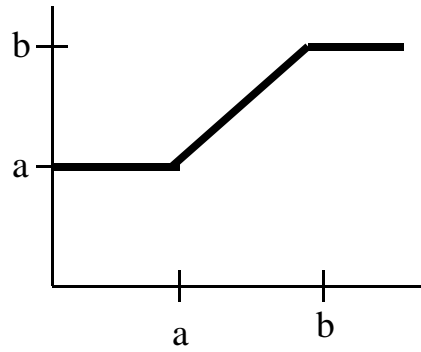


$\text{pulse}(a,b,x) = \text{step}(a,x) - \text{step}(b,x)$

Johns Hopkins Department of Computer Science  
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



## Clamp

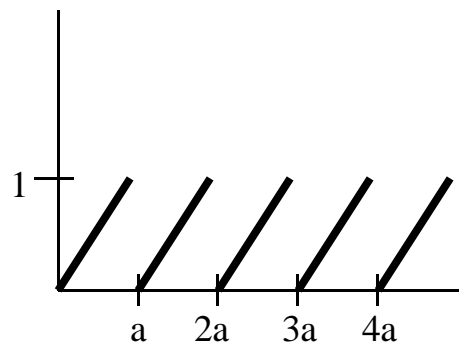


$$\text{clamp}(x,a,b) = \min(\max(x,a), b)$$

Johns Hopkins Department of Computer Science  
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



## Mod

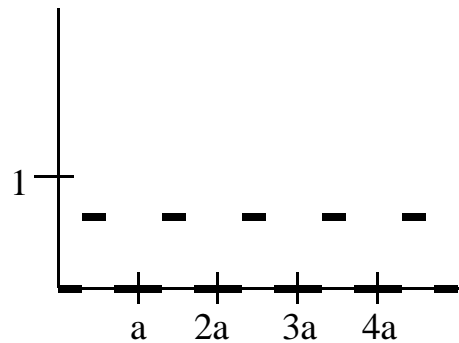


$$\text{mod}(x,a) / a$$

Johns Hopkins Department of Computer Science  
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



## Periodic Pulse



`pulse( 0.4, 0.6, mod(x,a)/a )`

Johns Hopkins Department of Computer Science  
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



## Example 1 - brick (see handout)

**Brick is primarily a 2D pulse**

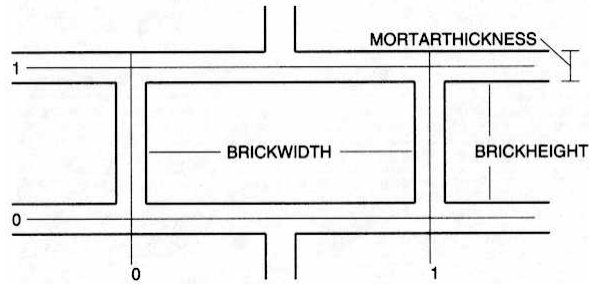
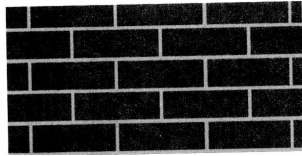
**Input parameters may include:**

- color of brick and mortar
- size of brick
- thickness of mortar
- mortar bump size
- frequency of brick color variation
- etc.

Johns Hopkins Department of Computer Science  
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



## Brick



from Ebert, ed., *Texturing and Modeling: a Procedural Approach*, 1994, pages 37-38.

Johns Hopkins Department of Computer Science  
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen



## Example 2 - star (see handout)

**Exploit symmetry of star geometry**

**Input parameters may include:**

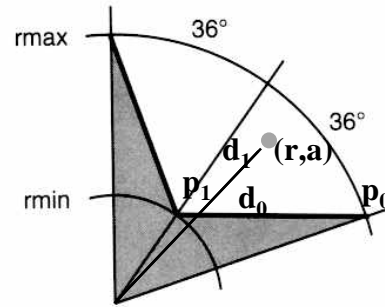
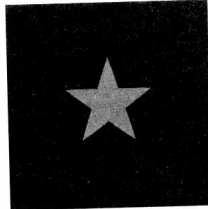
- **Inner and outer star radii**
- **Number of points**
- **Star and background colors**
- **Star bump parameters**
- **Parameters for star distribution**

Johns Hopkins Department of Computer Science  
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen





# Star



from Ebert, ed., *Texturing and Modeling: a Procedural Approach*, 1994, pages 44-46.

Johns Hopkins Department of Computer Science  
Course 600.456: Rendering Techniques, Professor: Jonathan Cohen

## Rendering Techniques Handout – Brick and Star Shaders

```
#include "proctext.h"

#define BRICKWIDTH      0.25
#define BRICKHEIGHT    0.08
#define MORTARTHICKNESS 0.01

#define BMWIDTH        (BRICKWIDTH+MORTARTHICKNESS)
#define BMHEIGHT      (BRICKHEIGHT+MORTARTHICKNESS)
#define MWF            (MORTARTHICKNESS*0.5/BMWIDTH)
#define MHF            (MORTARTHICKNESS*0.5/BMHEIGHT)

surface
brick(
    uniform float Ka = 1;
    uniform float Kd = 1;
    uniform color Cbrick = color (0.5, 0.15, 0.14);
    uniform color Cmortar = color (0.5, 0.5, 0.5);
)
{
    color Ct;
    point Nf;
    float ss, tt, sbrick, tbrick, w, h;
    float scoord = s;
    float tcoord = t;

    Nf = normalize(faceforward(N, I));

    ss = scoord / BMWIDTH;
    tt = tcoord / BMHEIGHT;

    if (mod(tt*0.5,1) > 0.5)
        ss += 0.5; /* shift alternate rows */
    sbrick = floor(ss); /* which brick? */
    tbrick = floor(tt); /* which brick? */
    ss -= sbrick;
    tt -= tbrick;
    w = step(MWF,ss) - step(1-MWF,ss);
    h = step(MHF,tt) - step(1-MHF,tt);

    Ct = mix(Cmortar, Cbrick, w*h);

    /* diffuse reflection model */
    Oi = Os;
    Ci = Os * Ct * (Ka * ambient() + Kd * diffuse(Nf));
}

#include "proctext.h"

surface
star(
    uniform float Ka = 1;
    uniform float Kd = 1;
    uniform color starcolor = color (1.0000,0.5161,0.0000);
```

```

uniform float npoints = 5;
uniform float sctr = 0.5;
uniform float tctr = 0.5;
)
{
point Nf = normalize(faceforward(N, I));
color Ct;
float ss, tt, angle, r, a, in_out;
uniform float rmin = 0.07, rmax = 0.2;
uniform float starangle = 2*PI/npoints;
uniform point p0 = rmax*(cos(0),sin(0),0);
uniform point p1 = rmin*
    (cos(starangle/2),sin(starangle/2),0);
uniform point d0 = p1 - p0;
point d1;

ss = s - sctr; tt = t - tctr;
angle = atan(ss, tt) + PI;
r = sqrt(ss*ss + tt*tt);
a = mod(angle, starangle)/starangle;

if (a >= 0.5)
    a = 1 - a;
d1 = r*(cos(a), sin(a),0) - p0;
in_out = step(0, zcomp(d0^d1));
Ct = mix(Cs, starcolor, in_out);

/* diffuse ("matte") shading model */
Oi = Os;
Ci = Os * Ct * (Ka * ambient() + Kd * diffuse(Nf));
}

```