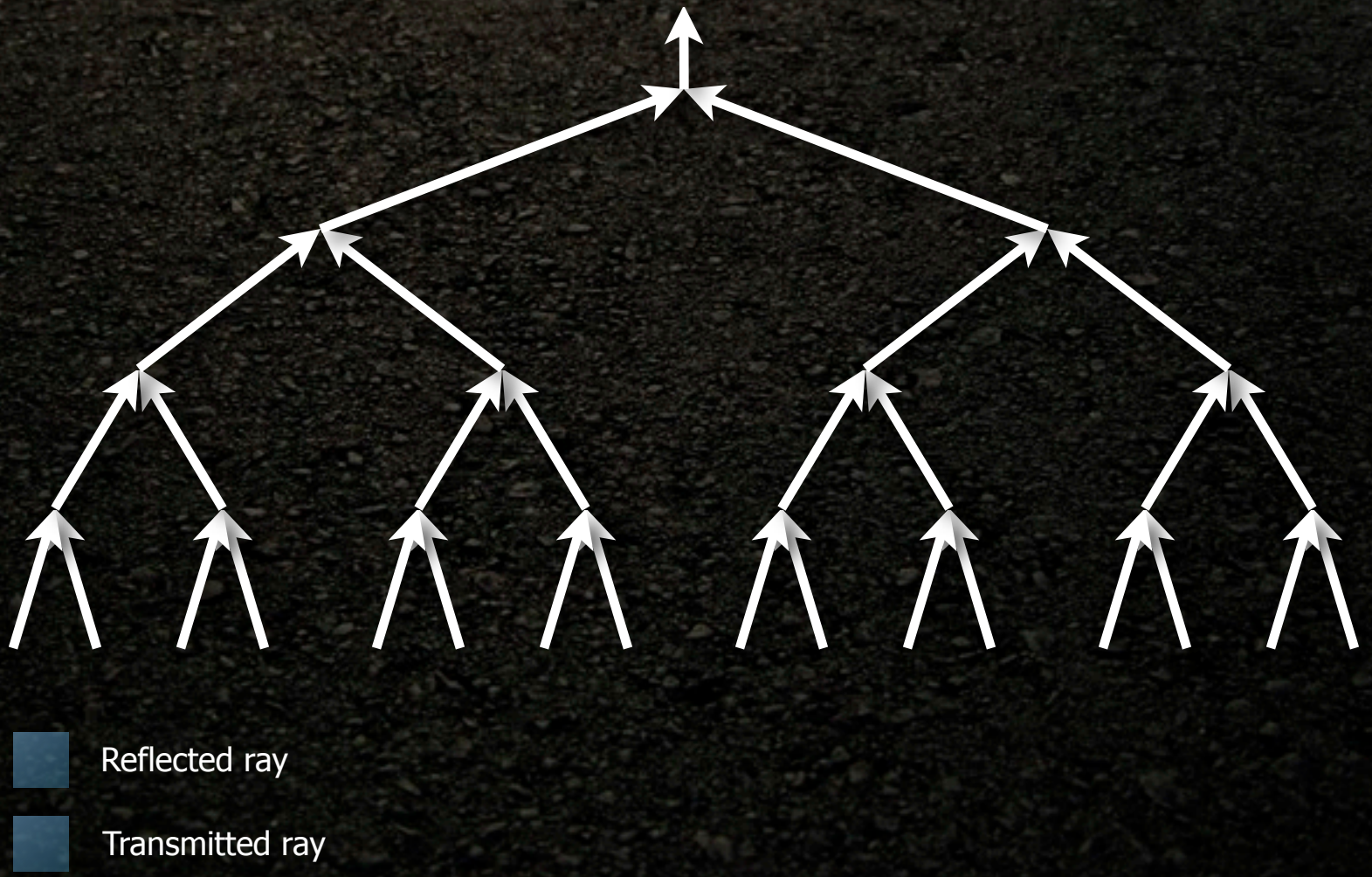

CS 6620
Texture 1
NVIDIA.

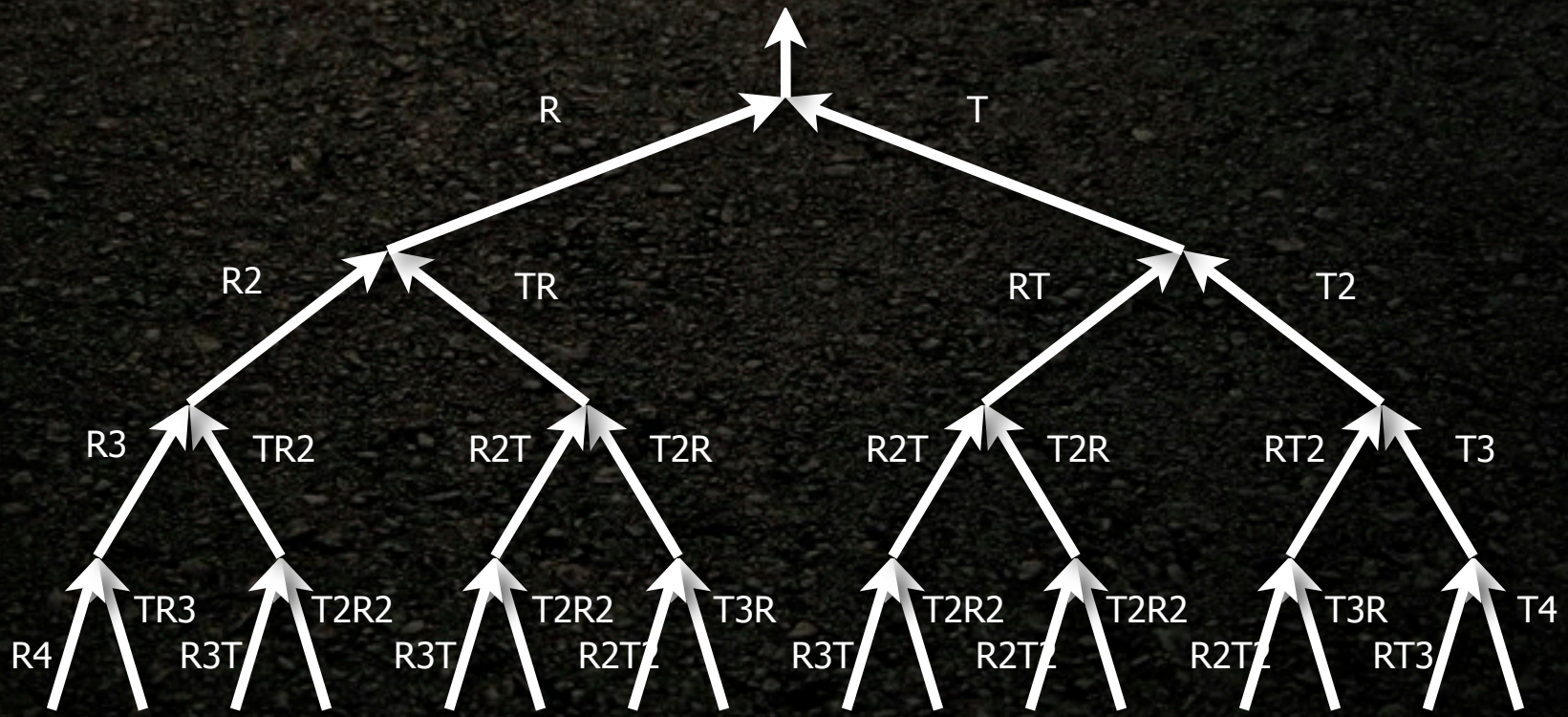
Ray trees



Reflected ray

Transmitted ray

Ray trees


R
T
R2
TR
RT
T2
R3
TR2
R2T
T2R
R2T
T2R
RT2
T3
R4
TR3
R3T
T2R2
R3T
T2R2
T3R
R3T
T2R2
T2R2
T3R
T4
R2T2
R2T2
RT3

Pruning ray trees

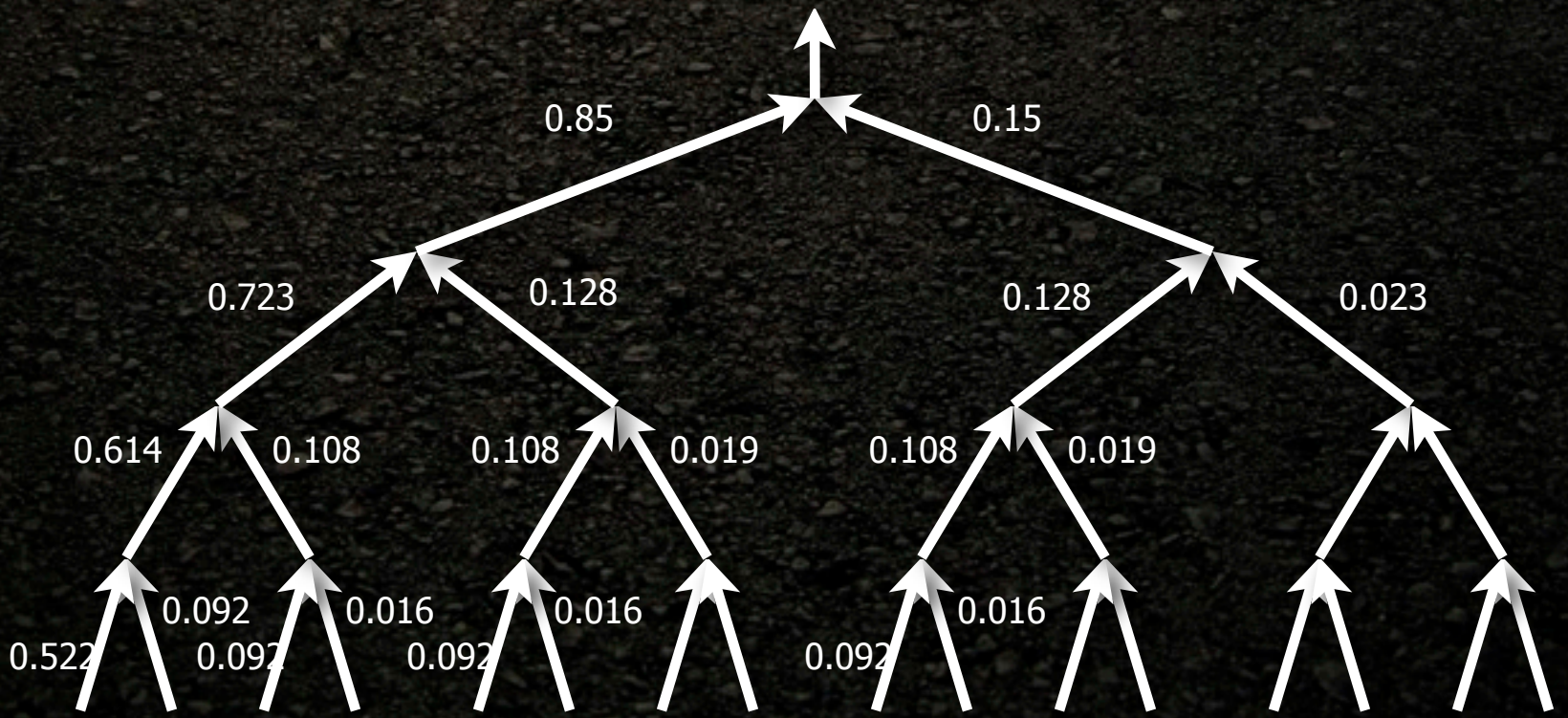
Mitigate explosion of rays by terminating the recursion at some minimum attenuation threshold:

```
if (depth >= max_depth || atten < min_atten)
  result = Color(0, 0, 0)
else
  result = traceRay(…)
```

Requires passing attenuation down the ray tree as you traverse

Pruning ray trees

Reflectance = 0.85
Transmittance = 0.15
Min. attenuation = 0.1

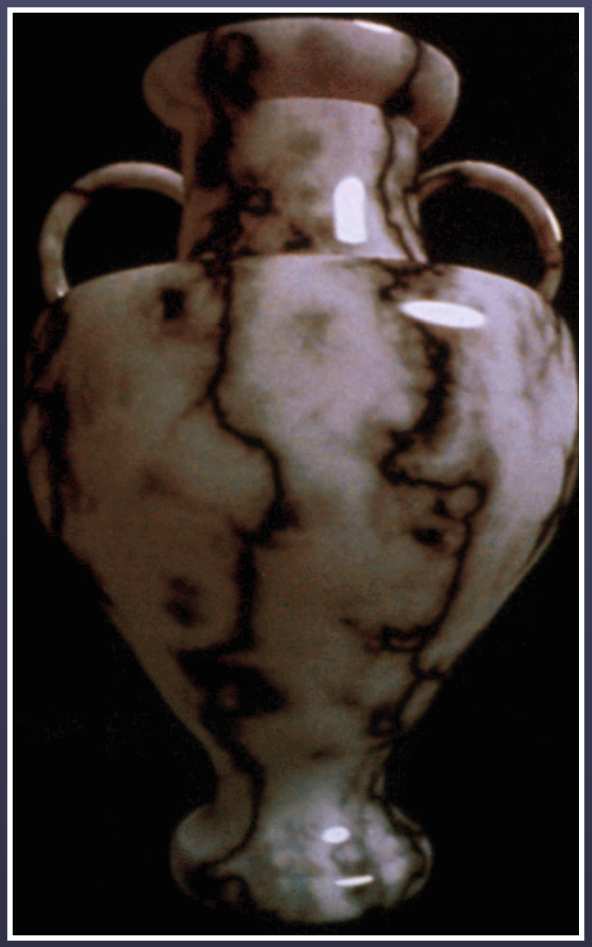


Texture mapping

- Most real objects do not have uniform color
- Texture: color and other material properties as a function of space

Texture mapping topics

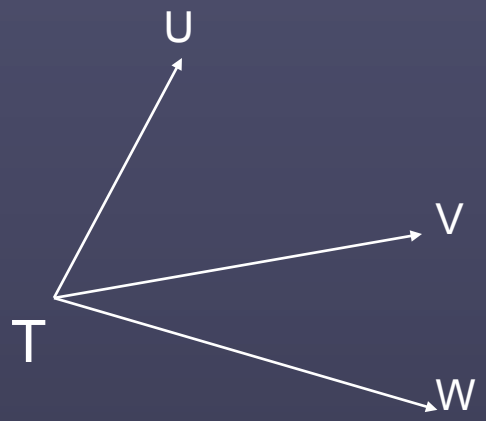
- Image textures
- Texture coordinates
  - Linear, cylindrical, spherical mappings
  - Barycentric coordinates for triangles
  - Software architecture
- Procedural textures
  - Simple: checkerboards, tiles, etc.
  - Fractal noise based: marble, granite, wood, etc.
- Bump mapping



Ken Perlin, NYU

Linear projection

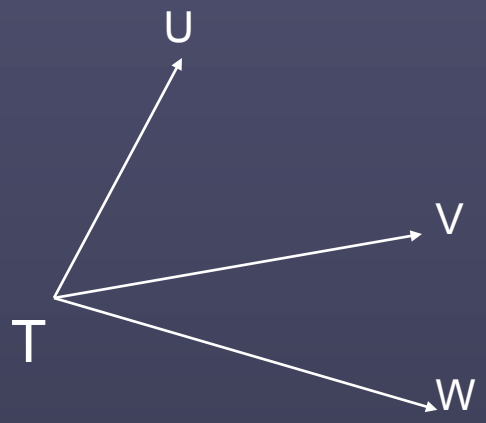
$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} U_x & V_x & W_x \\ U_y & V_y & W_y \\ U_z & V_z & W_z \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$$



inverse mapping:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} U_x & V_x & W_x \\ U_y & V_y & W_y \\ U_z & V_z & W_z \end{bmatrix}^{-1} \begin{bmatrix} x - T_x \\ y - T_y \\ z - T_z \end{bmatrix}$$

Linear special case



$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} + \begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix}$$

inverse mapping:

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} x - T_x \\ y - T_y \\ z - T_z \end{bmatrix}$$

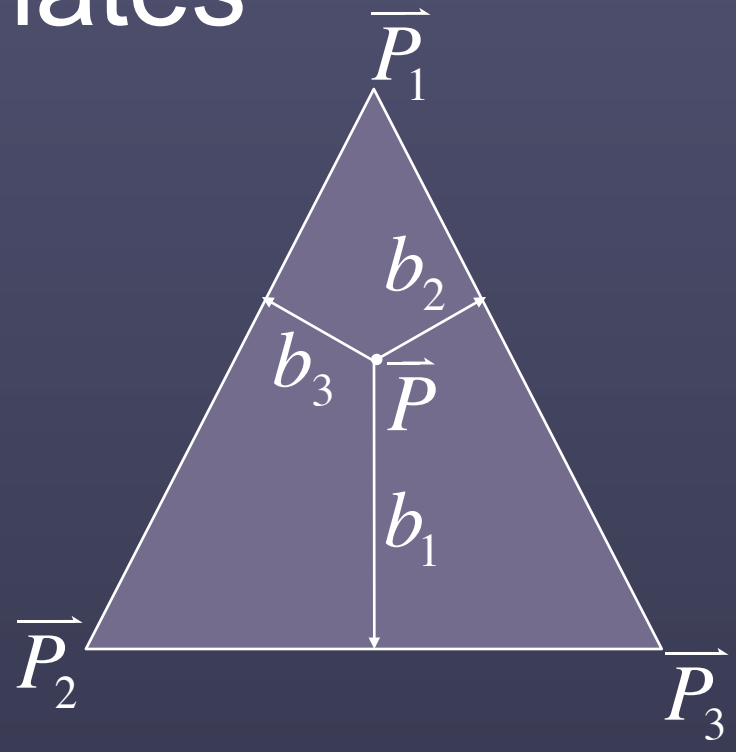
Barycentric coordinates

$$0 \le b_1, b_2, b_3 \le 1$$

$$b_1 + b_2 + b_3 = 1$$

$$\overrightarrow{P} = b_1\overrightarrow{P_1} + b_2\overrightarrow{P_2} + b_3\overrightarrow{P_3}$$

$$= b_1\overrightarrow{P_1} + b_2\overrightarrow{P_2} + (1 - b_1 - b_2)\overrightarrow{P_3}$$



$$u = b_1$$

$$v = b_2$$

$$w = 0$$

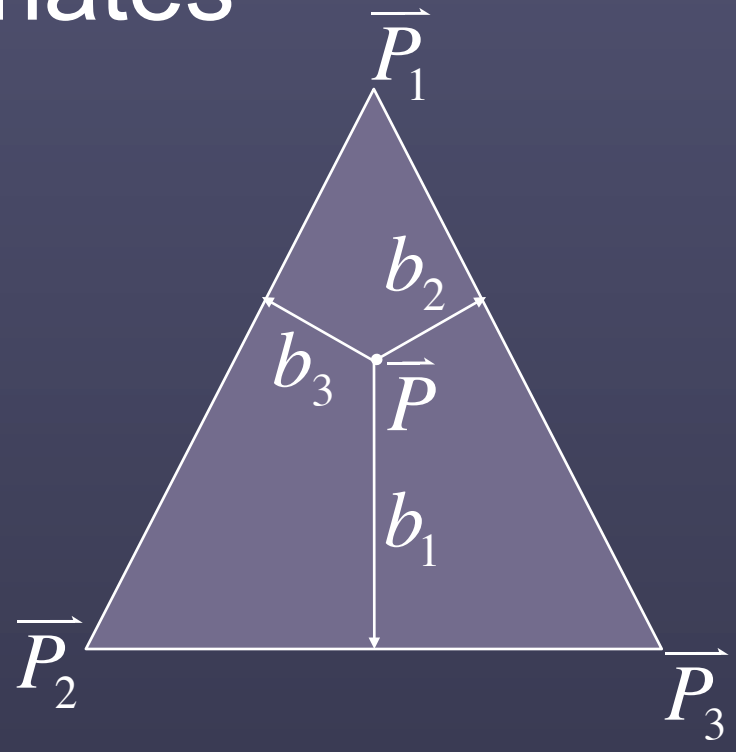
Specified texture coordinates

$$0 \le b_1, b_2, b_3 \le 1$$

$$b_1 + b_2 + b_3 = 1$$

$$\overrightarrow{P} = b_1\overrightarrow{P_1} + b_2\overrightarrow{P_2} + b_3\overrightarrow{P_3}$$

$$= b_1\overrightarrow{P_1} + b_2\overrightarrow{P_2} + (1 - b_1 - b_2)\overrightarrow{P_3}$$



$$\mathrm{u} = b_1 u_1 + b_2 u_2 + (1 - b_1 - b_2) u_3$$

$$v = b_1 v_1 + b_2 v_2 + (1 - b_1 - b_2) v_3$$

$$w = b_1 w_1 + b_2 w_2 + (1 - b_1 - b_2) w_3$$

Other coordinate transforms

- Any of those spherical projections
- Could transform u/v on the disc or ring (similar to cylindrical)
- Conical projections
- Many more

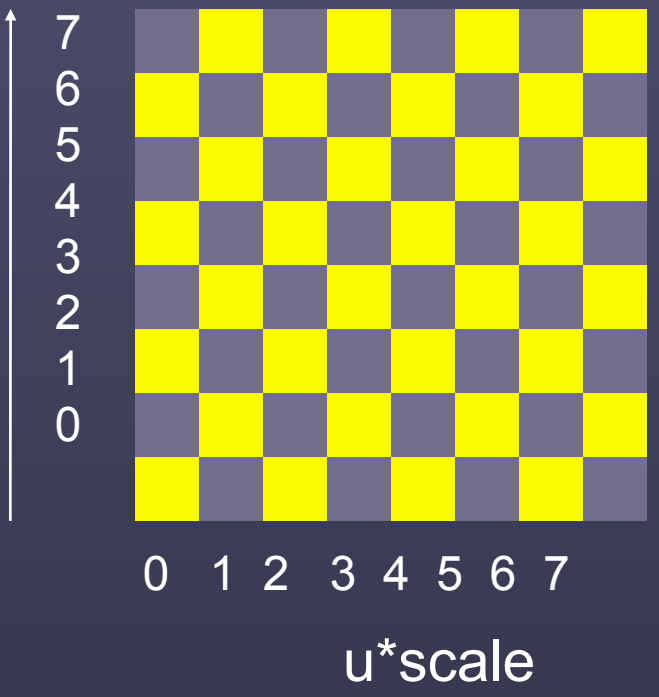
Checkerboard texture

$$i_1 = (\text{int})(u * scale)$$
$$i_2 = (\text{int})(v * scale)$$

$$cell = (i1 + i2)\%2$$

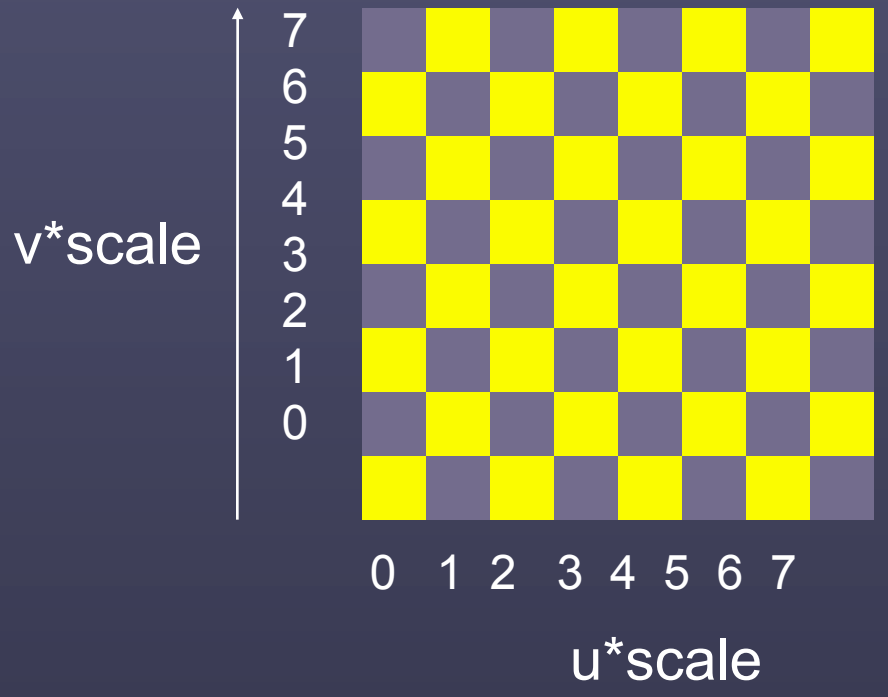
$if\,(cell = 0)$ use color1

else use color2



3D Checkerboard texture

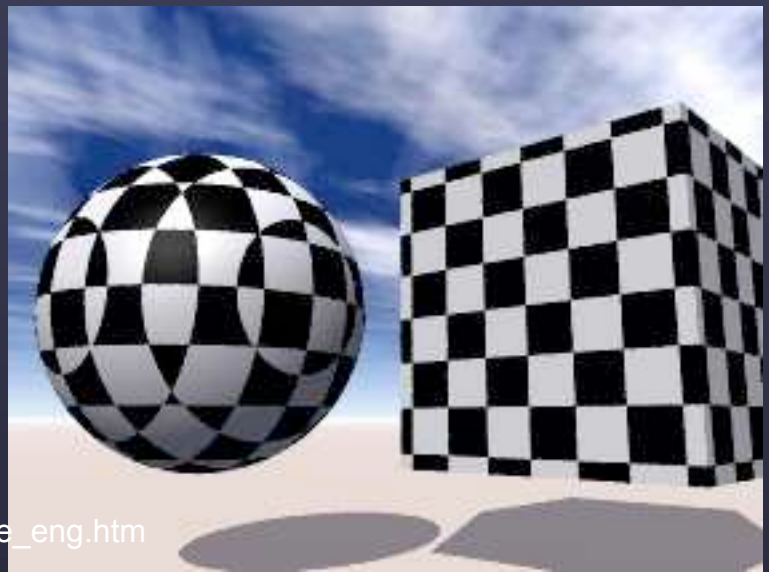
$$i_1 = (\text{int})(u * scale)$$
$$i_2 = (\text{int})(v * scale)$$
$$i_3 = (\text{int})(w * scale)$$



$$cell = (i1 + i2 + i3)\%2$$

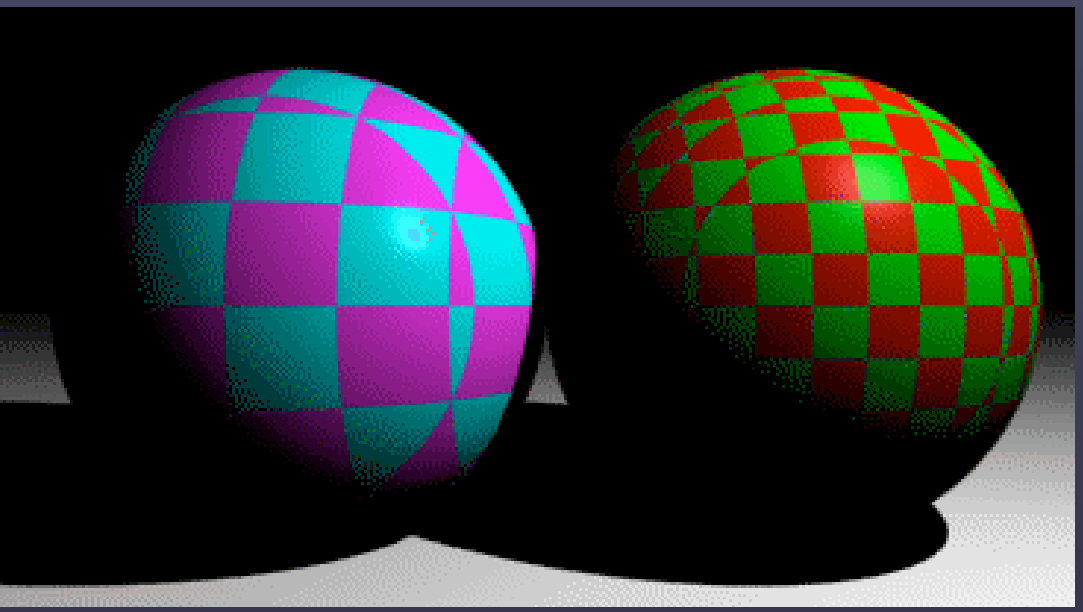
$if\ (cell = 0)$ use color1

else use color2

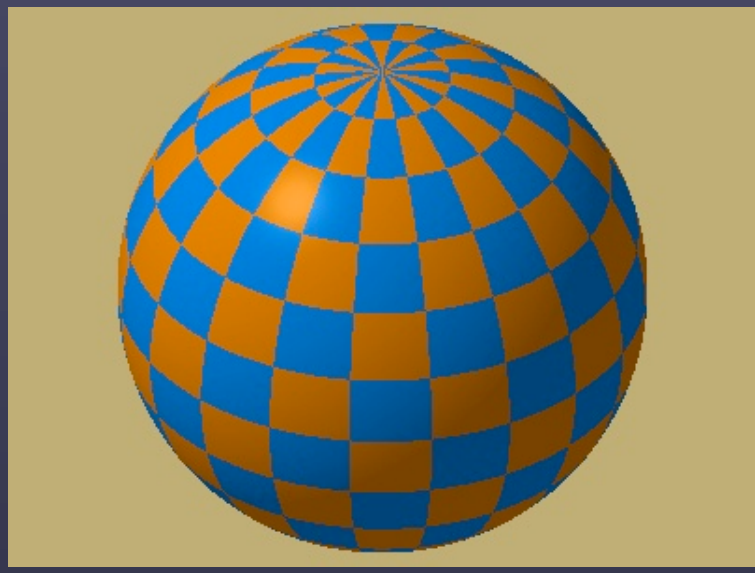


http://www.chez.com/jrlivenais/vdesprit/tut_index/check_sphere/check_sphere_eng.htm

Checkboard comparisons



3D checker
planar mapping



2 D (or 3D) checker

spherical mapping

http://cgg.ms.mff.cuni.cz/~pepca/lectures/textures/sample/checker.jpg

http://courses.dce.harvard.edu/~cscie234/projects/Slocum/checker-spheres-4x4-jitt-1.gif

Floor tile texture

$$s = u * scale - (\text{int})(u * scale)$$

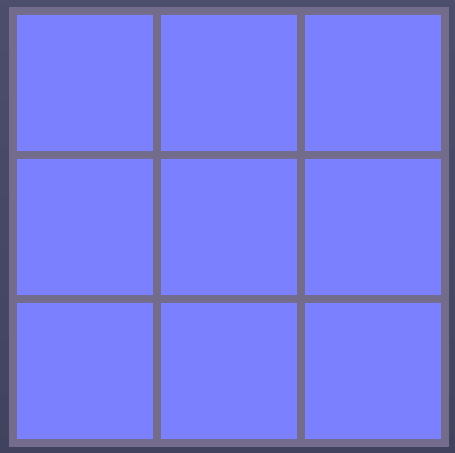
$$t = v * scale - (\text{int})(v * scale)$$

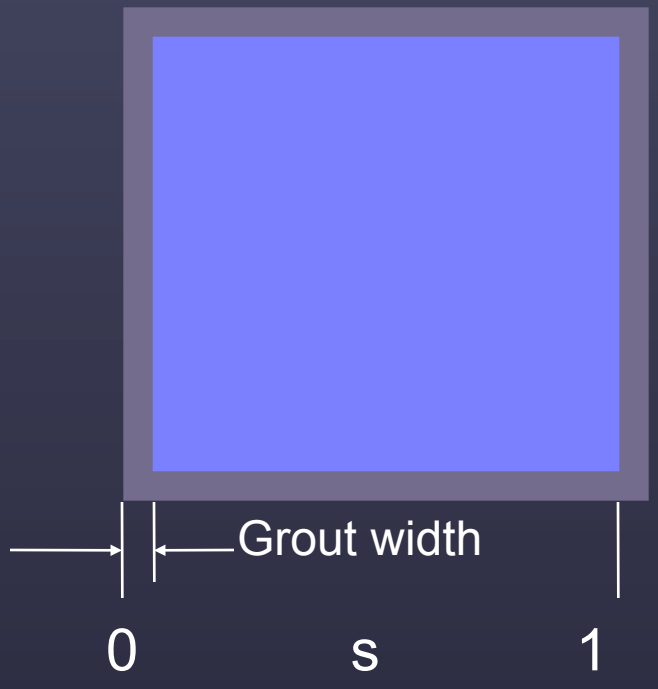
$$if\,(s < grout\_width \;\|\; t < grout\_width)$$

$$\quad color = grout\ color$$

$$else$$

$$\quad color = tile\ color$$





Brick texture

$ti = (\text{int})(v * vscale)$ // get brick row

$u' = u * uscale - (ti\%2) * 0.5$ // shift column for odd rows

$s = u' - (\text{int})u'$ // brick column fraction

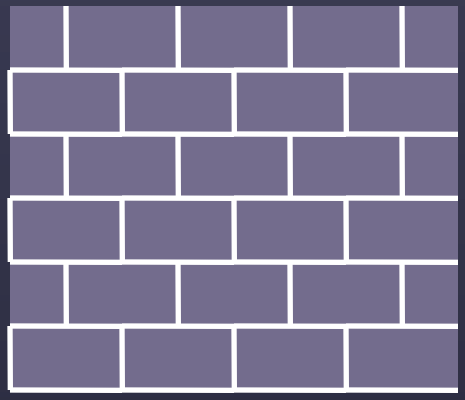
$t = v * vscale - ti$ // brick row fraction

$if\,(s < mortar\_width \,||\, t < mortar\_width)$

$color = mortar\_color$

$else$

$color = brick\_color$



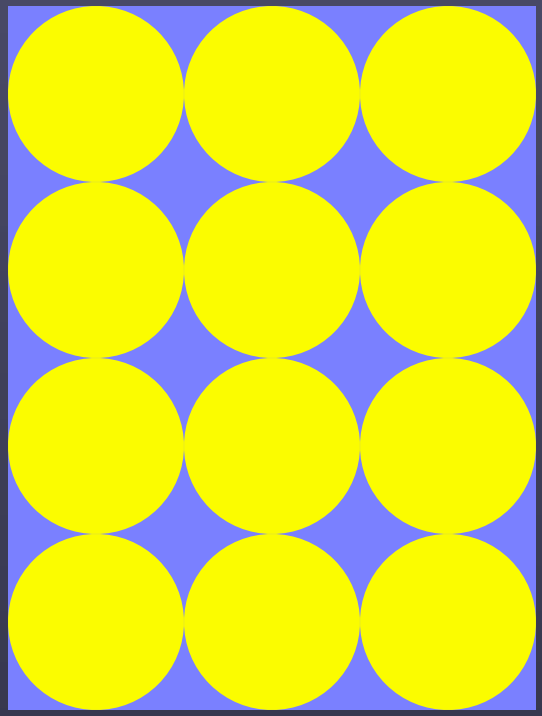
More flexible implementation

- Instead of choosing between two (or more) colors, choose between two materials

```
if(cell == 0)
   matl1->shade(…);
else
   matl2->shade(…);
```

Other simple procedural textures

- Hexagonal mapping
- Cutout shapes
- Use your imagination!

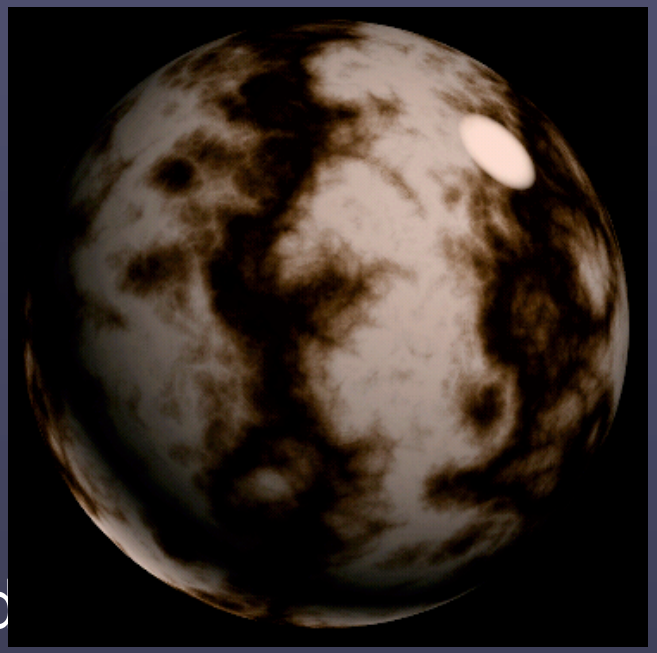


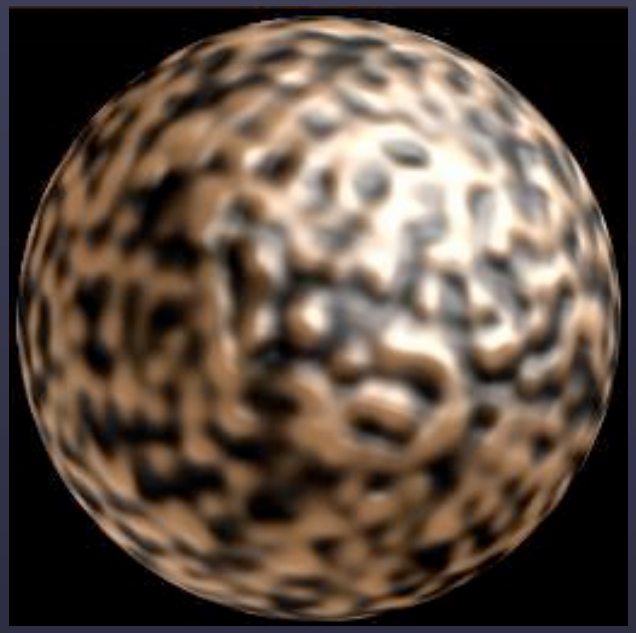
Solid textures

- Many objects are created out of solid materials
- Texture is three-dimensional
- Surface texture is cutaway of the 3D texture space
- Examples:
  - Wood
  - Marble
  - Granite

Perlin noise

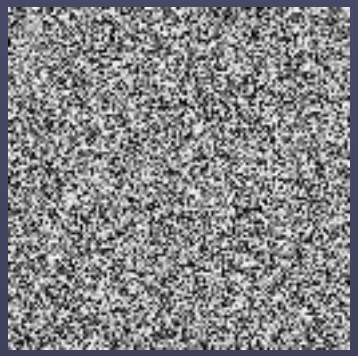
- A pseudorandom method for making natural textures
- Appeared in Siggraph 1985 (preview in 1984)
- Ken Perlin won an academy award (technical achievement award) in 1997
- More info: http://noisemachine.com
- Improved in 2002: http://mrl.nyu.edu/~perlin/noise





Perlin noise

- Need a function that:
  - Looks random
  - Has fixed frequency content
  - Is coherent (value changes smoothly from one point to another)
- Solution: smooth random values on a regular grid



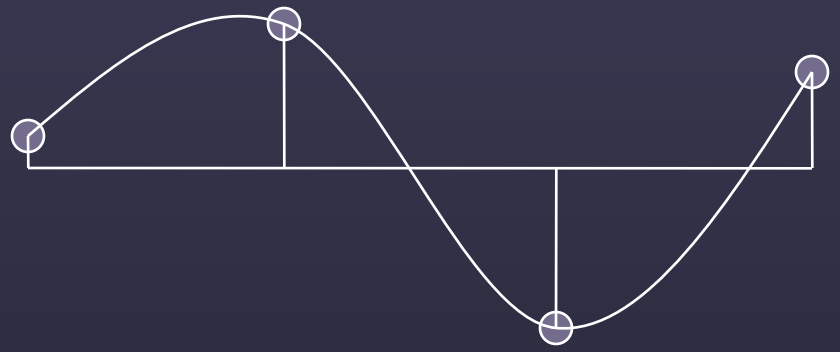
Incoherent

Coherent

http://www.robo-murito.net/code/perlin-noise-math-faq.html

Value noise function

- Idea: place random values on a grid lattice
- Interpolate between them (spline)
- Spline is expensive in 3D



Perlin noise (gradient noise)

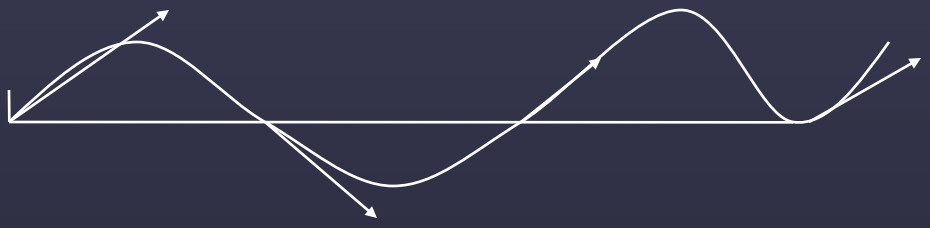
- Idea: place random gradients on a grid lattice
- Weighted interpolation of gradient values

$$0 \le x \le 1$$

$$g_0 = G_0 x, g_1 = G_1 (1 - x)$$

$$noise(x) = lerp(g_0, g_1, ease(x))$$

Note: $noise(0) = 0, noise(1) = 0$



Questions?