cheap tricks

- texture mapping
- bump mapping
- procedural texture mapping

procedural texture mapping

- procedure returns a texture color for any point in 3d space (note this is not an image stack)
- sample to find texture for surface

procedural textures

- advantages
  - don't need to find a mapping from a (complex) 3d surface to a 2d texture image
  - concise representation of texture
- disadvantages
  - ad hoc techniques cannot duplicate photographs

perlin noise - 1D example

step 1: generate discrete noise function with specified length, amplitude, sampling frequency

example: length=8, amplitude = 3, sampling frequency is 7 Hz

seed \rightarrow \text{random number generator} \rightarrow r_0, r_1, \ldots, r_7 \in [0,1]

1 sec \rightarrow 3r_3
perlin noise – 1D example

step 2: interpolate with smoothing

perlin noise – 1D example

step 3: repeat with various amplitudes/frequencies

step 4: add together

perlin noise – 2D example

for more info see Perlin Noise link on proj2 web site

cheap tricks

• texture mapping
• bump mapping
• procedural texture mapping
• jittering

jittering: antialiasing technique

Run rt on Solaris for example

jittering: anti-aliasing technique

for pixel i,j:
  • cast several random rays through pixel i,j region into scene
  • for each:
    • find intersection point (if any) that is closest to eye
    • compute color at intersection
    • compute average of all samples as color for pixel i,j
random ray

- Shoot ray \( P_{ij} \) through \( P_{ij} + (dx, dy) \) where
- \( dx \) is chosen uniformly at random over \([-W/(2w), W/(2w)]\)
- \( dy \) is chosen uniformly at random over \([-H/(2h), H/(2h)]\)

cheap tricks

- texture mapping
- bump mapping
- procedural texture mapping
- jittering
- soft shadows

soft shadows

In reality, we don't have point lights (or point spot lights)!

- cast several shadow rays at random points on "light"
- use \#occlusions/\#rays as shadow weight

run rt on solaris for examples

soft shadows

For directional lights

- cast several shadow rays perturbed by small vector in tangent plane
- use \#occlusions/\#rays as shadow weight

ray tracing

- simple ray casting
- recursive ray tracing
- modeling transforms
- cheap tricks
- optimizations
**ray tracing complexity**

\[ O(\text{# of intersection tests}) = O(\text{#pixels \times # objects}) \]

Can we reduce the number of intersection tests?

**optimization**

- bounding boxes
- oct-trees
- BSP-trees

**bounding boxes: intuition**

- most rays miss the object

**bounding boxes**

1. rule out rays by simple intersection test with bounding box
2. perform exhaustive test on remaining rays

**scene graph**

- body assembly
- body
- head assembly
- head
- eye 1
- eye 2

Each node has an axis-aligned bounding box:
- test ray against box - if no intersection return alpha=-1
- otherwise proceed as usual

**constructing the bounding boxes**

- Leaf of scene graph
  - Sphere: radius \( r \), center \( c=(c_x, c_y, c_z) \)
    - Corners of bounding box are \( (c_x-r, c_y-r, c_z-r), (c_x+r, c_y+r, c_z+r) \)
  - Triangle: vertices \( (x_0, y_0, z_0), (x_1, y_1, z_1), (x_2, y_2, z_2) \)
    - Corners of bounding box are \( \)
constructing bounding box

child provides parent the corners of its bounding "box" in parent's coordinate system
i.e. \( M_{p0}, M_{p1}, \ldots, M_{p7} \) where \( M \) transforms the child into its parent coordinate system

parent computes

- "extrema" of coordinates in each dimension: \( x_{\text{min}}, x_{\text{max}}, y_{\text{min}}, y_{\text{max}}, z_{\text{min}}, z_{\text{max}} \)
- bounding corners
  \( p(x,y,z) \) where \( x(x_{\text{min}}, x_{\text{max}}), y(y_{\text{min}}, y_{\text{max}}), z(z_{\text{min}}, z_{\text{max}}) \)

bounding box with corners
\( p_0, p_1, \ldots, p_7 \)

bounding box intersection

1. find intersection of ray with each bounding plane of box

\[ \begin{align*}
\text{front plane intersection:} \\
q &= p + \alpha v \\
q &= (p_x + \alpha v_x, p_y + \alpha v_y, p_z + \alpha v_z) \\
\text{where} \quad \alpha &= (z_{\text{max}} - p_z) / v_z
\end{align*} \]

bounding box intersection test

if ray intersects with node's bounding box:
- check for intersection with local object
- pass ray to children
otherwise return no intersection

REMEmBER: bounding box encloses objects at node and all bounding boxes of children

optimization

- bounding boxes
- oct-trees
- BSP-trees

We won't cover these now. Look on the web for details.