EECS 487
December 4, 2006

- Rodney Ma presentation
- precomputed radiance transfer (cont'd)
- project 5 concepts
support code

• not jot
• written by Prof. Guskov
• command line raytracer:
  srt scene.sce rendering.tga
• “srt” = simple ray tracer
• scene: lights, camera, objects, ...
• output: targa image (see spec)
tasks

1. ray generation code
2. shading computations
3. interpolated normals for smooth shading
4. specular reflections
5. cylinder primitive
6. anti-aliasing
7. area lights
8. optimization: bounding sphere test
ray generation

• same camera model we've seen before

• parameters:
  \( \mathbf{e} \): eye location
  \( \mathbf{u} \): unit vector pointing right
  \( \mathbf{v} \): unit vector pointing up
  \( \mathbf{w} \): unit vector pointing behind us
  rendering window width, height in pixels
  field of view angle (in vertical direction)
  distance to near plane
ray generation
ray generation

- image maps to rectangle in near plane
- assume center of rectangle is (0,0)
- Q: what are (l, b) in terms of (r, t)?
ray generation

- pixel (0,0) is *center* of lower left pixel
- Q: what are coordinates at corners?
ray generation

• convert pixel coordinates \((i,j)\) to \((u,v)\) coordinates describing location on near clipping plane
• e.g. \((-\frac{1}{2}, -\frac{1}{2})\) in pixels maps to \((l,b,n)\) in eye coordinates
  \((n = w \text{ coordinate of near plane})\)
• world-space location \(s\) is then:
  \[ s = e + uu + vv + wn \]
• Q: what is the ray?
ray generation

• Q: what is the ray?
• A: $r(t) = \mathbf{e} + t(\mathbf{s} - \mathbf{e})$
ray generation

• Q: how to get l, r, t, b, n, f?
ray generation

• Q: how to get l, r, t, b, n, f?
• A: given n, and fovy
  \[ \theta = \text{fovy}/2 \]
ray generation

- \( \tan(\theta) = \frac{t}{n} \)
- solve for \( t \)
ray generation

- aspect ratio $a = \text{image width}/\text{height}$
- $r = a \cdot t$
/// Returns the color from the shading computation using
/// the information in the hitinfo_t structure
/// level is the recursion level
XVecf RayTracerT::Shade(const hitinfo_t & hit, int level) {
    XVecf color(0.0f);

    // Ambient light contribution
    color = hit.m_mat.m_ca*hit.m_mat.m_cr;

    // YOUR CODE HERE
    // shading code here
    // iterate over the lights and collect their contribution
    // make a recursive call to Trace() function to get the reflections

    return color;
}
/// Returns the color from the shading computation using
/// the information in the hitinfo_t structure
/// level is the recursion level
XVecf RayTracerT::Shade(const hitinfo_t& hit, int level) {
    XVecf color(0.0f);

    // Ambient light contribution
    color = hit.m_mat.m_ca*hit.m_mat.m_cr;

    // YOUR CODE HERE
    // shading code here
    // iterate over the lights and collect their contribution
    // make a recursive call to Trace() function to get the reflections

    SceneT::LightCt::const_iterator li;
    for(li=m_scene.BeginLights(); li!=m_scene.EndLights(); ++li) {
        ...
    }
    return color;
}
phong shading

• do per pixel normals
  – use barycentric coordinates (provided)
  – return interpolated normal within mesh triangle in `MeshT::Intersect()`
  – if: `m_shade == PHONG_SHADE`
specular reflections

• if max recursion not reached, trace a reflection ray to compute reflected color
• compute illumination seen along reflected array
• combine with base color using material specular value
remaining tasks...

- cylinder primitive
- anti-aliasing
- area lights
- optimization: bounding sphere test