

Real-Time Rendering

Mike Gleicher
November 15, 2005
These are notes – not projected in class

Realism and Rendering

- Rendering – creating images
- Realism
 - Philosophical issues (what is real)
 - Photorealism
 - Indistinguishable from a photograph of the same things
- Non-Photorealistic (intentionally stylized)

Two Paths to Realistic Rendering

Primitives Based

- Build on fast abstractions
 - Polygon rendering
 - Texture Mapping
- Hacks to get visual effects
- What we can do fast today
- Moving Target (as hardware evolves)

Light Based

- Model the transport of light
- Scene geometry is used in computation
- Simulate the real lighting from the real world
- Better simulations give better effects
- Not based on convenient implementations

A Light-Based Approach: Ray-Tracing

- Model photons (rays of light) bouncing around
- Start at light, towards eye
 - Realistic, but VERY slow
- Start at eye, work backwards toward light
 - Not as realistic can be efficient
- Real physics requires cleverness
 - Lots of photons, how to do it?

Some observations on ray tracing

- Need to have the complete scene
- Irregular computations on complex data
- Next lecture will discuss light-based rendering

Primitive Rendering

- Can draw triangles really fast in hardware
- Can do simple local lighting really fast
- Can do texture mapping really fast
 - Fancier variants of texture mapping possible
- What hacks can you do to use these pieces to achieve more effects
- They ARE hacks!

Reminder: Texture Mapping

- Define “texture coordinates” for objects
- Change object properties per-pixel
- Use “Maps” (images) to look up values
- Use procedures to define maps
 - (less common)
- Most common: texture map chooses color

What effects to add?

- More complex surfaces
- Reflections
- Shadows
- More complex lighting
- ...
- All using building blocks of texture mapped polygons and local lighting

How to make a “textured” surface?

- Coloring doesn’t really make brick ...
 - Surfaces have “micro-shape”
 - Effects the light
 - Too hard to model with Polygons
- Fake with a Normal Map (or bump Map)
 - Per-pixel change of normal vector
 - Only changes normal vector (and therefore lighting)
- Doesn’t change geometry
 - No shadows, occlusions, silhouettes, ...

Beyond Bump Maps

- Displacement Maps
 - Actually effect surface – move the point
 - Should change the normal too
 - Gives occlusions, ...
- Problem: harder to do
 - Must be done before visibility computation
 - Can’t rasterize in-order

Lightmaps

- How to get cooler lighting effects?
- Paint them onto the surfaces
 - Put dark splotches for shadows
 - Put bright spots for where lights go
- Use multiple layers of textures
 - Blend them together (multiply)
 - Add them on top of each other
- Important extension to primitive set:
 - Multi-texture, multi-pass

More Lightmaps

- Animated Lightmaps
- Using Volume textures as lights (position in world) to get local effects
- Get spotlights and other kinds of shapes
- Get effects that are computed offline

Multipass Rendering



- Draw objects multiple times to accumulate effects
 - Simple: get more than 4 light sources
 - Complex: get many kinds of textures
- Use tricks to partially draw objects
 - Stencil buffers
- Multi-textures
 - Multi-passes can only layer, not multiply
 - E.g. a light source attenuates a texture

An Aside: Other sources of texture coords



- Decal Textures (world space)
 - Label on can
- Projector Textures
 - Slide projector
 - Coordinates are image plane of projector
 - We're not dealing with occlusions (yet)
- Boundaries of texture space
 - Repeat, clamp, mirror, ...

Can we do lighting this way? (with occlusions = shadows)



- Only want to light front object
 - Or dim back objects
- Use Z-Buffer for visibility!
- Render from Light's point of view
- Use as a projector texture
 - Tells where light hits
 - Check to make sure pixel matches map
 - Can use Z, or object ID – issues in comparison
- Shadow Map

How about reflections?



- Render from virtual eyepoint
 - Works for flat mirrors
 - Rarely done
- Assume small object / big world
 - Paint world onto an environment map
 - Sphere / Cube
 - Color of point depends looks up in environment map

Environment Mapping



- Texture coordinates depend on eye and normal
- Simplification: assume object is small relative to work
 - All points at origin (since the object's size is negligible)
 - All viewing directions the same (since object is so small)
 - Outgoing ray depends only on surface normal
 - Compute texture coordinate based on normal

What a bag of hacks!



- Game / interactive graphics programmers keep making more!
- Each generation of hardware gives new features, leads to new tricks
- More and more effects
- Isn't there a more principled way?
 - Yes, its to model light!