3D Graphics Continued

Chris GauthierDickey
CIS 399 - Intro to Game Programming
Lighting and Shading

• How do we model lights and the shading that occurs naturally in a 3D environment?
  – Model light in the most basic terms:
    • surfaces either emit light, like the surface of the sun
    • or they reflect light, like a white wall
  – When we look at a surface, any point is determined by multiple light sources and reflective surfaces
Global Illumination Models

- Two basic global illumination models have been created:
  - Ray tracing: We extend our simple model of light and cameras.
    - In order for a point on an object to be illuminated, a ray of light must have reflected off it from a light source to our camera.
    - Ray tracing simply traces the path backwards from the camera to the object, reflects off of multiple objects, and checks to see if it intersected with a light source.
Ray Tracing Continued

• Shadows are created by casting shadow rays from the initial point intersected by our ray.
  – The shadow rays are aimed at the light sources and if they intersect an object, we recurse

• Transparency occurs by bending rays according to material properties
Radiosity

• Ray tracing doesn’t take diffuse light reactions into consideration
  – Simple example: light reflected off the diffuse surface of a green couch will tint a white carpet green

• Divide the scene into $n$ patches, calculate radiosity between each pair: $O(n^2)$ operation unfortunately.
Simplified Model

- We only consider the rays of light from the camera reflecting off the object towards the light sources (again, reverse rays).
- We ignore intervening objects
- We assume lights have a 3 component intensity: $I = [I_r, I_g, I_b]^T$. Intensities are 0-1 in red, green, blue (sometimes 0 - 255).
Ambient Sources

- We approximate those diffuse-diffuse interactions of light through ambient light sources. This source is simply represented by its three components:
  \[ I_a = [ I_{ar} \ I_{ag} \ I_{ab} ]^T. \]
- Every point in the scene receives this amount of light
Point Light Sources

• The ideal point source radiates light equally in all directions. The closer you are to it, the more light you receive.
  – $I(P_0) = \begin{bmatrix} I_{\text{r}}(P_0) & I_{\text{g}}(P_0) & I_{\text{b}}(P_0) \end{bmatrix}^T$

• Intensity at point $P$ is determined by:
  – $I(P, P_0) = \frac{1}{(\text{length of } P - P_0)^2} I(P_0)$
Distant or Directional Light Sources

• When a light source is far away, the light rays we receive are quite close to being parallel
  – Example: Sun or moonlight
  – Represented by a simple vector for faster light calculations
Spotlights

• Spotlights are lights that originate at a point and have an angle through which light emanates.
  – Represented by initial point, direction vector, and angle light radiates from initial point to direction vector.
  – Intensity may also drop off as light spreads away from direction vector towards angle
Lights in Ogre

• Ogre provides 3 types of lights
  – Point (LT_POINT)
  – Spotlight (LT_SPOTLIGHT)
  – Directional (LT_DIRECTIONAL)

• We create lights with the SceneManager
  – light = mSceneMgr->createLight("mylight")
Lights Continued

// now determine what kind of light it is
light->setType(Light::LT_POINT);
light->setPosition(Vector3(0, 150, 250));
// you can set the diffuse and specular colors
light->setDiffuseColor(1.0, 0.0, 0.0);
light->setSpecularColor(1.0, 0.0, 0.0);

// directional lights are created as so
light = mSceneMgr->createLight("myDirectionalLight");
light->setType(Light::LT_DIRECTIONAL);
light->setDirection(Vector3(0, -1, 1));
More Lights

// spotlights
light = mSceneMgr->createLight("mySpotlight");
light->setType(Light::LT_SPOTLIGHT);
light->setDirection(-1, -1, 0);
light->setPosition(Vector3(0, 0, 0));
// spotlights have a brighter inner core than outer beam, so we
// set this spotlight to have a 35 degree inner beam, and 50 degree outer
// beam as so:
light->setSpotlightRange(Degree(35), Degree(50));
Modeling the World

• For the next project, we will need to model our virtual world. What do we need?
• Let’s imagine space is divided into sectors
  – Each sector will have its own unique skybox
  – Each sector has planets located in various places
Entities

• Recall that Ogre uses Entities to model things that we can render to the screen
• We, on the other hand, want to consider objects that exist in our world...
  – space ships
  – planets
  – asteroids
  – missiles
Our World

- Space
  - contains Sectors
    - contains Objects (Mobile or Static)
      - Each frame, we update mobile objects
    - Objects take an Entity and SceneNode pointer, so they can update the scene
More Modeling

• Ships are mobiles
  – Ships have weapon array
  – A selected weapon type
  – Shield level (and max level)
  – Armor level (and max level)
  – Energy level (and max level)
  – Max velocity
  – Velocity and Direction

• Weapons are mobiles
  – have velocity, direction, damage amount
Input with OGRE

• Two basic types of input
  – Unbuffered Input
  – Buffered Input

• Unbuffered Input simply checks the state of the mouse and keyboard when you call its functions

• Buffered Input is more event driven
  – Appears that it will change once again!
Unbuffered Input

• Take a look at ExampleFrameListener.h
  – We need 2 things
    • PlatformManager, to get an input reader
    • InputReader, to capture the state of hardware
  – Then we poll from InputReader:
    • call capture()
    • call isKeyDown(KeyCode kc), getMouse*

• Note that this kind of reading is dependent on the framerate
Buffered Input

• For buffered input, we need two things:
  – EventProcessor: to act as a queue for events
  – RenderWindow: to initialize the event processor (call ep->initialize(render_window))

• We also need to create KeyListeners, MouseListeners, MouseMotionListeners
  – These respond to the appropriate events
Responding to Events

• If we use BufferedInput, how should we respond to events?
  – Events should call methods on the class we are controlling with the input method
    • space-bar calls ship->thrust(), for example

• What about animation?
  – Each entity should have an update method, that lets the class figure out how to be updated.