Realtime Rendering

ICS 162 Lecture #16
Realism versus Realtime

**Academic:** Simulate transport of light around a scene and interaction with materials

**Game Industry:** Way too slow 😞 need lots of engineering work to make things interactive / realtime while maintaining realism
Today: Realtime Graphics

• Graphics Rendering Pipeline
  – Efficiency in the rendering pipeline

• Lighting and Shadow
  – Remember lighting?
  – Tricks with texture mapping
  – Environment mapping
  – Light maps
  – Bump maps and displacement maps
  – Shadows
Classic 3D Rendering Pipeline

3D Primitives

- **Modeling Transformation**
- **Lighting**
- **Viewing Transformation**
- **Projection Transformation**
- **Clipping**
- **Viewport Transformation**
- **Scan Conversion**

1. **Transform** into 3D world coordinate system
2. Illuminate according to lighting and reflection
3. **Transform** into 3D camera coordinate system
4. **Transform** into 2D camera coordinate system
5. Clip primitives outside camera’s view
6. **Transform** into image coordinate system
7. **Draw pixels** (includes texturing, hidden surface, …)
GPU Fundamentals: The Graphics Pipeline

- A simplified graphics pipeline
  - Note that pipe widths vary, many caches etc. not shown
GPU Fundamentals: The Modern Graphics Pipeline

Key developments

- Multiple passes of rendering
- Render-to-texture is a key “feedback loop” useful for many computations
GPU Pipeline: Transform

• Vertex Processor
  – Transform from “world space” to “image space”
  – Compute per-vertex lighting
Transformations

$p(x,y,z)$

3D Object Coordinates

Modeling Transformation

3D World Coordinates

Viewing Transformation

3D Camera Coordinates

Projection Transformation

2D Screen Coordinates

Viewport Transformation

2D Image Coordinates

$p'(x',y')$

Transformations map points from one coordinate system to another
Viewing Transformations

- **3D Object Coordinates**
- **3D World Coordinates**
- **3D Camera Coordinates**
- **2D Screen Coordinates**
- **2D Image Coordinates**
GPU Pipeline: Rasterizer

- **Rasterizer**
  - Convert geometric rep. (vertex) to image rep. (fragment)
    - **Fragment** = image fragment
      - Pixel + associated data: color, depth, stencil, etc.
    - Interpolate per-vertex quantities across pixels
GPU Pipeline: Shade

• Fragment Processors
  – Compute a color for each pixel based on fragment colors, texture, stencil, blending etc.
Today: Realtime Graphics

• Graphics Rendering Pipeline
  – Efficiency in the rendering pipeline

• Lighting and Shadow
  – Remember lighting?
  – Tricks with texture mapping
  – Environment mapping
  – Light maps
  – Bump maps and displacement maps
  – Shadows
Efficiency in graphics pipeline

• Streaming Geometry Formats
  – How we will deliver the geometry stream to the graphics subsystem
    • Vertices, normals, texture coordinates, colors
    • e.g. 3 floats per vertex, 2 floats per texture, 3 floats per normal, 3 floats per color
      ➔ V3f T2f N3f C3f ➔ 132 bytes per triangle
  – Careful geometry packing can yield 2x gains in performance

• Visibility Determination
  – Clipping, Culling and Occlusion Testing

• Resolution Determination
  – Level of Detail (LOD) analysis
Visibility culling
(Clipping & Culling)

View frustum culling (=clipping)

Occlusion culling

Back-face culling
Resolution Determination

• Examples: huge mountains and thousands of trees
  – Clipping (aperture of 60) $\Rightarrow$ 1/6 of the total triangle
  – Culling $\Rightarrow$ 1/12 (= $1/2 \times 1/6$)
  – Occlusion $\Rightarrow$ about 1/15
– Geometry
  • Terrain
    – 20km*20km square terrain patch with sampled every meter
    $\Rightarrow$ 400 million triangle map
  • Trees
    – Realistic tree $\Rightarrow$ 25,000 triangle
    – One tree for every 20 meters $\Rightarrow$ 25 billion triangle per frame ??
Level-of-detail rendering

• use different levels of detail at different distances from the viewer
Level-of-detail rendering

- not much visual difference, but a lot faster

- use area of projection of BV to select appropriate LOD
Resolution Determination

- Multiresolution
  - Human visual system tends to focus on larger, closer object (Especially if they are moving)
  - Two components
    - A resolution-selection (heuristic)
      - The distance from the object to the viewer
        » Far from perfect (The object is far away, but huge??)
      - The area of the projected polygon
        » Perceived size, not with distance
    - Rendering algorithms that handles the desired resolution
      - A discrete approach (memory intensive) ➔ (Noticeable popping)
        » Simply select the best-suited model from a catalog of varying quality models
      - A continuous method (high CPU hit?)
        » Derive a model with the right triangle count on the fly

Clipping, Culling, and occlusion tests determine what we are seeing
Resolution test determine how we see it
Today: Realtime Graphics

• Graphics Rendering Pipeline
  – Efficiency in the rendering pipeline

• Lighting and Shadow
  – Remember shading?
  – Tricks with texture mapping
  – Environment mapping
  – Light maps
  – Bump maps and displacement maps
  – Shadows
Diffuse Reflection (Lambertian)

The greater the angle between the normal and the vector from the point to the light source, the less light is reflected. Most light is reflected when the angle is 0 degrees, none is reflected at 90 degrees.
Specular Reflection
(Phong Lighting Model)

• Maximum specular reflectance occurs when the viewpoint is along the path of the perfectly reflected ray (when \( \alpha \) is zero).
• Specular reflectance falls off quickly as \( \alpha \) increases.
• Falloff approximated by \( \cos^n(\alpha) \) where \( n \) controls the size of the specular lobe.
Reflectance Model

\[ R = k_a I + k_d I \max (\hat{\mathbf{l}} \cdot \hat{\mathbf{n}}, 0) + k_s I \max (\hat{\mathbf{r}} \cdot \hat{\mathbf{v}}, 0)^p \]

What do we need to know to compute the color of a vertex with this model?
Simple shading lacks realism

• Can’t simulate effects of complicated geometry, materials and light sources efficiently
  – global illumination effects: soft shadows, inter-reflections, etc.
  – complicated light paths: reflective and translucent materials
Today: Realtime Graphics

• Graphics Rendering Pipeline
  – Efficiency in the rendering pipeline

• Lighting and Shadow
  – Remember lighting?
  – Tricks with texture mapping
  – Environment mapping
  – Light maps
  – Bump maps and displacement maps
  – Shadows
Surface Normals

Flat Shading – each polygon face has a normal that is used to perform lighting calculations.
Texture Mapping

• The problem
  – Colors, normals, etc. are only specified at vertices.
  – How do we add detail between vertices?

• Solution
  – Specify the details in an image (the texture) and specify how to apply the image to the geometry (the map)

• Works for shading parameters other than color, as we shall see
  – The basic underlying idea is the UV mapping
UV-Mapping

$(x,y,z) \leftrightarrow (u,v)$
Packing Textures
Packing Textures

• Hardware works most efficiently when texture dimensions are a power of 2.

• Problem
  – The limits on texture width/height make it inefficient to store many textures
  – For example: long, thin objects

• Solution
  – Artists pack the textures for many objects into one image
    • The texture coordinates for a given object may only index into a small part of the image
    • Care must be taken at the boundary between sub-images to achieve correct blending
    • Difficulty: how to resample packed textures?
Some effects are easier to implement if multiple textures can be applied.
Texture Matrix

• Normally, the texture coordinates given at vertices are interpolated and directly used to index the texture

• The texture matrix applies a general (perspective) transformation to the texture coordinates before indexing the texture

• What use is this?
Animating Texture

• Loading a texture onto the graphics card is very expensive
• But once there, the texture matrix can be used to “transform” the texture
  – For example, changing the translation can select different parts of the texture
• If the texture matrix is changed from frame to frame, the texture will appear to move on the object
• This is particularly useful for things like flame, or swirling vortices, or pulsing entrances, …

(see ferris wheel model in zotlandia)
Projective Texturing

• Projecting a texture onto the scene, as if from a slide projector
• Using a texture matrix:
  – Equate texture coordinates with world coordinates
  – Think about it from the projector’s point of view: wherever a world point appears in the projector’s view, it should pick up the texture
  – Use a texture matrix equivalent to the projection matrix for the projector – maps world points into texture image points
Today: Realtime Graphics

• Graphics Rendering Pipeline
  – Efficiency in the rendering pipeline

• Lighting and Shadow
  – Remember lighting?
  – Tricks with texture mapping
  – Environment mapping
  – Light maps
  – Bump maps and displacement maps
  – Shadows
Environment Mapping

- Environment mapping produces reflections on shiny objects
- Texture is transferred in the direction of the reflected ray from the environment map onto the object
- Reflected ray: $R = 2(N \cdot V)N - V$
- What is in the map?
Environment Maps

• The environment map may take one of several forms:
  – Cubic mapping
  – Spherical mapping (two variants)
  – Parabolic mapping

• Describes the shape of the surface on which the map “resides”

• Determines how the map is generated and how it is indexed

• What are some of the issues in choosing the map?
Environment Mapping

- Cube mapping is the norm nowadays
Environment Maps

Miller and Hoffman, 1984
Environment Mapping

www.debevec.org

Need For Speed Underground

Far Cry
Planar Reflections (Flat Mirrors)

• Basic idea:
  – We need to draw all the stuff around the mirror
  – We need to draw the stuff in the mirror, reflected, without drawing over the things around the mirror

• Key point: You can reflect the viewpoint about the mirror to see what is seen in the mirror, or you can reflect the world about the mirror
Reflecting Objects

- If the mirror passes through the origin, and is aligned with a coordinate axis, then just negate appropriate coordinate
- More generally, transform into mirror space, reflect, transform back
Reflection Example

- The stencil buffer after the second pass
- The color buffer after the second pass – the reflected scene cleared outside the stencil
Reflection Example

The color buffer after the final pass
Today: Realtime Graphics

• Graphics Rendering Pipeline
  – Efficiency in the rendering pipeline

• Lighting and Shadow
  – Remember lighting?
  – Tricks with texture mapping
  – Environment mapping
  – **Light maps**
  – Bump maps and displacement maps
  – Shadows
Light Sources

• Ambient
  – basic, even illumination of all objects in a scene

• Directional
  – all light rays are in parallel in 1 direction - like the sun

• Point
  – all light rays emanate from a central point in all directions – like a light bulb

• Spot
  – point light with a limited cone and a fall-off in intensity – like a flashlight

Penumbra angle
(light starts to drop off to zero here)
Global Illumination

Nearly every surface in a scene acts as a light source!
Light Maps

• Speed up lighting calculations by pre-computing lighting and storing it in maps
  – Allows complex illumination models to be used in generating the map (e.g., shadows, radiosity)
  – Used in complex rendering algorithms (*Radiance*), not just games

• Issues:
  – How is the mapping determined?
  – How are the maps generated?
  – How are they applied at run-time?
Applying Light Maps

• Use multi-texturing hardware
  – First stage: Apply color texture map
  – Second stage: Modulate with light map
What type of lighting (diffuse, specular, reflections) can the map store?
Example

No light maps

With light maps
Fog Maps

- Dynamic modification of light-maps
- Put fog objects into the scene
- Compute where they intersect with geometry and paint the fog density into a dynamic light map
  - Use same mapping as static light map uses
- Apply the *fog map* as with a light map
  - Extra texture stage
Fog Map Example
Today: Realtime Graphics

• Graphics Rendering Pipeline
  – Efficiency in the rendering pipeline

• Lighting and Shadow
  – Remember lighting?
  – Tricks with texture mapping
  – Environment mapping
  – Light maps
  – Bump maps and displacement maps
  – Shadows
Bump Mapping

- Texture values perturb surface normals

Stores heights: can derive normals
Dot Product bump mapping

• Store normal vectors in the bump map
• Apply the bump map using the dot3 operator
  – Takes a dot product
Normal Mapping

original mesh
4M triangles

simplified mesh
500 triangles

simplified mesh
and normal mapping
500 triangles

DOOM 3

James Hastings-Trew
Environment Bump Mapping

- Perturb the environment map lookup directions with the bump map
• Bump Mapping Problems
  – Doesn’t take into account geometric surface depth
    • Does not exhibit parallax
    • No self-shadowing of the surface
    • Coarse silhouettes expose the actual geometry being drawn

• Displacement Mapping
  – Displace actual positions from Heightfield Map
Mesh refinement on the GPU vertex processor

Only work with coarse mesh on the CPU
Today: Realtime Graphics

• Graphics Rendering Pipeline
  – Efficiency in the rendering pipeline

• Lighting and Shadow
  – Remember lighting?
  – Tricks with texture mapping
  – Environment mapping
  – Light maps
  – Bump maps and displacement maps
  – Shadows
Why Shadows?

- Shadows tell us about the relative locations and motions of objects
Facts about Shadows

• Shadows can be considered as areas hidden from the light source
  – Suggests the use of hidden surface algorithms

• For scenes with static lights and geometry, the shadows are fixed
  – Can pre-process such cases and use lightmaps
  – Cost is in moving lights or objects

• Point lights have hard edges, and area lights have soft edges
Soft-shadows
Shadows in Light Maps

• Static shadows can be incorporated into light maps
  – When creating the map, test for shadows by ray-casting to the light source - quite efficient

• Area light sources should cast soft shadows
  – Interpolating the texture will give soft shadows, but not good ones, and you lose hard shadows
  – Sampling the light will give better results: Cast multiple rays to different points on the area light, and average the results
  – Should still filter for best results
Soft Shadow Example
Shadows in Games

Megaman

You can do a bunny hop by holding down the L1 button then releasing it. The longer you hold the button the higher the jump.

Grand Theft Auto

Metal Gear
Ground Plane Shadows

- Shadows cast by point light sources onto planes are an important case that is relatively easy to compute
  - Shadows cast by objects (cars, players) onto the ground
Projective Shadows

• Create a texture (dark on white) representing the appearance of the occluder as seen by the light
  – Game programmers frequently call this a *shadow map*
  – Can create it by “render to texture” with the light as viewpoint

• Use *projective texturing* to apply it to receivers

• Requires work to identify occluders and receivers
Phase 1: Render from Light

- Depth image from light source
Phase 1: Render from Light

- Depth image from light source
Phase 2: Render from Eye

- Standard image (with depth) from eye
Phase 2+: Project to light for shadows

- Project visible points in eye view back to light source

(Reprojected) depths match for light and eye. VISIBLE
Phase 2+: Project to light for shadows

- Project visible points in eye view back to light source

(Reprojected) depths from light, eye not the same. BLOCKED!!
Visualizing Shadow Mapping

- A fairly complex scene with shadows

*the point light source*
Visualizing Shadow Mapping

- The scene from the light’s point-of-view
Visualizing Shadow Mapping

- The depth buffer from the light’s point-of-view
Visualizing Shadow Mapping

- Projecting the depth map onto the eye’s view

FYI: depth map for light’s point-of-view again
Visualizing Shadow Mapping

- Comparing light distance to light depth map

Green where the light planar distance and the light depth map are equal

Non-green is where shadows should be
Hardware Shadow Map Filtering

GL_NEAREST: blocky  GL_LINEAR: antialiased edges

Low shadow map resolution reveals artifacts
Shadow Volumes
Shadow Volumes

• A *shadow volume* for an object and light is the volume of space that is shadowed
  – That is, all points in the volume are in shadow for that light/object pair
• Creating the volume:
  – Find *silhouette edges* of shadowing object as seen by the light source
  – *Extrude* these edges away from the light, forming polygons
  – Clip the polygons to the view volume
Example: Doom 3
Projective vs. Shadow Volumes

• Projective: Relatively fast, doesn’t require generating any new geometry

• Volumes: Can create pixel accurate shadows, not limited by texture memory
What does the Renderer do?

First render phase
- Shadow pass: Cascaded Shadows for day light, night-time shadows, cloud shadows, character shadows
- Z pre-pass / G-Buffer update / Light Pre-pass

Second render phase
- Screen-space ambient occlusion / local irradiance

Third render phase
- In case of a Light Pre-Pass renderer: fill up the light buffer with all light properties
- Global Illumination: collects light data
- Reflections / refractions / environment reflections

Fourth render phase (Opaque objects)
- Lighting and rendering the opaque objects with shadow data sorted by shaders and / or front-to-back (also water + refraction map)

Fifth render phase
- Dynamic sky-dome

Sixth render phase (Transparent objects)
- Lighting and rendering the transparent objects with shadow data sorted back-to-front + alpha-to-coverage objects
- Foliage rendering

Seventh render phase (Particles)
- Render high-res particles into back-buffer
- Render low-res particles into smaller render target
- Composite smaller particle render target with main render target

Eighth render phase
- MSAA resolve to a potentially lower-res render target (for certain target platforms)
- PostFX (HDR, Depth of Field, Depth Fog, Height-Based fog, motion blur, heat haze, tear gas, drunk-effect

Ninth render phase
-- Up-scaling of lower res render target (for certain target platforms)
-- UI etc.
Today: Realtime Graphics

• Graphics Rendering Pipeline
  – Efficiency in the rendering pipeline

• Lighting and Shadow
  – Remember lighting?
  – Tricks with texture mapping
  – Environment mapping
  – Light maps
  – Bump maps and displacement maps
  – Shadows