Real-Time Rendering

COS 526: Advanced Computer Graphics
Motivation

- Today, can create photorealistic renderings
  - Complex geometry, lighting, materials, shadows
  - Computer-generated movies/special effects
    (difficult or impossible to tell real from rendered…)

- But algorithms are slow (minutes, hours, days)
Real-Time Rendering

- Goal: interactive rendering. Critical in many apps
  - Games, visualization, computer-aided design, ...

- Until 10-15 years ago, focus on complex geometry

- *Chasm between interactivity, realism*
Evolution of 3D graphics rendering

Interactive 3D graphics pipeline as in OpenGL

- Earliest SGI machines (Clark 82) to today
- Most of focus on more geometry, texture mapping
- Some tweaks for realism (shadow mapping, accum. buffer)
Offline 3D Graphics Rendering

Ray tracing, radiosity, photon mapping
- High realism (global illum, shadows, refraction, lighting,..)
- But historically very slow techniques

“So, while you and your children’s children are waiting for ray tracing to take over the world, what do you do in the meantime?” Real-Time Rendering

Pictures courtesy Henrik Wann Jensen
New Trend: Acquired Data

- Image-Based Rendering: Real/precomputed images as input
- Also, acquire geometry, lighting, materials from real world
- Easy to obtain or precompute lots of high quality data. But how do we represent and reuse this for (real-time) rendering?
15 years ago

- High quality rendering: ray tracing, global illumination
  - Little change in COS 426 syllabus over the past 15 years
- Real-Time rendering: Interactive 3D geometry with simple texture mapping, fake shadows (OpenGL, DirectX)
- Complex environment lighting, real materials (velvet, satin, paints), soft shadows, caustics often omitted in both

- *Realism, interactivity at cross purposes*
Today: Real-Time Game Renderings

Unreal Engine 4

https://www.youtube.com/watch?v=gtHamLNPXyk#t=33

Digital Ira: NVIDIA, USC
Today

- Vast increase in CPU power, modern instrs (SSE, Multi-Core)
  - Real-time raytracing techniques are possible
    (even on hardware: NVIDIA Optix)

- 4th generation of graphics hardware is programmable
  - (First 3 gens were wireframe, shaded, textured)
  - Modern NVIDIA, ATI cards allow vertex, fragment shaders

- Great deal of current work on acquiring and rendering with realistic lighting, materials…

- *Focus on quality of rendering, not quantity of polygons, texture*
Goals

- Overview of basic techniques for high-quality real-time rendering

- Survey of important concepts and ideas, but do not go into details of writing code

- Some pointers to resources, others on web
Outline

- Motivation and Demos
- Programmable Graphics Pipeline
- Shadow Maps
- Environment Mapping
High quality real-time rendering

- Photorealism, not just more polygons
- Natural lighting, materials, shadows

Interiors by architect Frank Gehry. Note rich lighting, ranging from localized sources to reflections off vast sheets of glass.
High quality real-time rendering

- Photorealism, not just more polygons
- Natural lighting, materials, shadows

Real materials diverse and not easy to represent by simple parameteric models. Want to support measured reflectance.
High quality real-time rendering

- Photorealism, not just more polygons
- Natural lighting, materials, shadows

Naturol lighting creates a mix of soft diffuse and hard shadows.

- small area light, sharp shadows
  Agrawala et al. 00

- soft and hard shadows
  Ng et al. 03
Today: Full Global Illumination
Applications

- Entertainment: Lighting design
- Architectural visualization
- Material design: Automobile industry
- Realistic Video games
- Electronic commerce
Programmable Graphics Hardware (circa 2008)
Precomputation-Based Methods

- Static geometry
- Precomputation
- Real-Time Rendering (relight all-frequency effects)
- Involves sophisticated representations, algorithms
Video: Real-Time Relighting
Spherical Harmonic Lighting

Avatar 2010, based on Ramamoorthi and Hanrahan 01, Sloan 02
Interactive RayTracing

Advantages
- Very complex scenes relatively easy (hierarchical bbox)
- Complex materials and shading for free
- Easy to add global illumination, specularities etc.

Disadvantages
- Hard to access data in memory-coherent way
- Many samples for complex lighting and materials
- Global illumination possible but expensive

Modern developments: Leverage power of modern CPUs, develop cache-aware, parallel implementations

https://www.youtube.com/watch?v=kcP1NzB49zU
Sparse Sampling, Reconstruction

- Same algorithm as offline Monte Carlo rendering
- But with smart sampling and filtering (current work)
Outline

- Motivation and Demos
- *Programmable Graphics Pipeline*
- Shadow Maps
- Environment Mapping
Basic Hardware Pipeline

Application → Geometry → Rasterizer

CPU

Create geometry, lights, materials, textures, cubemaps, ... as inputs

Transform and lighting calcs. Apply per-vertex operations

GPU

Textures, Cubemaps

Per-pixel (per-fragment) operations
Geometry or Vertex Pipeline

These fixed function stages can be replaced by a general per-vertex calculation using vertex shaders in modern programmable hardware.
These fixed function stages can be replaced by a general per-fragment calculation using fragment shaders in modern programmable hardware.
OpenGL Rendering Pipeline

Vertices → Geometry Primitive Operations → Scan Conversion (Rasterize) → Fragment Operations

Images → Pixel Operations → Texture Memory

Programmable in Modern GPUs (Vertex Shader)

Programmable in Modern GPUs (Fragment Shader)

Framebuffer

Traditional Approach: Fixed function pipeline (state machine)
New Development (2003-): Programmable pipeline
Simplified OpenGL Pipeline

- User specifies vertices (vertex buffer object)

- For each vertex in parallel
  - OpenGL calls user-specified vertex shader: Transform vertex (ModelView, Projection), other ops

- For each primitive, OpenGL rasterizes
  - Generates a fragment for each pixel the fragment covers

- For each fragment in parallel
  - OpenGL calls user-specified fragment shader: Shading and lighting calculations
  - OpenGL handles z-buffer depth test unless overwritten

- Modern OpenGL is “lite”: basically just a rasterizer
  - “Real” action in user-defined vertex, fragment shaders
Shading Languages

- Vertex / Fragment shading described by small program
- Written in language similar to C but with restrictions
- Long history. Cook’s paper on Shade Trees, Renderman for offline rendering

- Stanford Real-Time Shading Language, work at SGI
- Cg from NVIDIA, HLSL
- GLSL directly compatible with OpenGL 2.0
Phong Shader: Vertex

This Shader Does
• Gives eye space location for v
• Transform Surface Normal
• Transform Vertex Location

```cpp
varying vec3 N;
varying vec3 v;

void main(void)
{
    v = vec3(gl_ModelViewMatrix * gl_Vertex); // Created For Use Within Frag Shader
    N = normalize(gl_NormalMatrix * gl_Normal);

    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
} // (Update OpenGL Built-in Variable for Vertex Position)
```
varying vec3 N;
varying vec3 v;

void main (void) 
{
    // we are in Eye Coordinates, so EyePos is (0,0,0)
    vec3 L = normalize(gl_LightSource[0].position.xyz - v);
    vec3 E = normalize(-v);
    vec3 R = normalize(-reflect(L,N));

    //calculate Ambient Term:
    vec4 lamb = gl_FrontLightProduct[0].ambient;

    //calculate Diffuse Term:
    vec4 Idiff = gl_FrontLightProduct[0].diffuse * max(dot(N,L), 0.0);

    // calculate Specular Term:
    vec4 Ispec = gl_FrontLightProduct[0].specular
        * pow(max(dot(R,E),0.0), gl_FrontMaterial.shininess);

    // write Total Color:
    gl_FragColor = gl_FrontLightModelProduct.sceneColor + lamb + Idiff + Ispec;
}
Outline

- Motivation and Demos
- Programmable Graphics Pipeline
- *Shadow Maps*
- Environment Mapping
Shadow and Environment Maps

- Basic methods to add realism to interactive rendering

- Shadow maps: image-based way hard shadows
  - Very old technique. Originally Williams 78
  - Many recent (and older) extensions
  - Widely used even in software rendering (RenderMan)
  - Simple alternative to raytracing for shadows

- Environment maps: image-based complex lighting
  - Again, very old technique. Blinn and Newell 76
  - Huge amount of recent work (some covered in course)

- Together, give most of realistic effects we want
  - But cannot be easily combined!!
  - See Annen 08 [real-time all-frequency shadows dynamic scenes] for one approach: convolution soft shadows
Common Real-time Shadow Techniques

Projected planar shadows

Light maps

Shadow volumes

Hybrid approaches

This slide, others courtesy Mark Kilgarc
Problems

Mostly tricks with lots of limitations

- Projected planar shadows
  works well only on flat surfaces

- Stenciled shadow volumes
  determining the shadow volume is hard work

- Light maps
  totally unsuited for dynamic shadows

- In general, hard to get everything shadowing everything
Shadow Mapping

- Lance Williams: Brute Force in image space (shadow maps in 1978, but other similar ideas like Z buffer, bump mapping using textures and so on)

- Completely image-space algorithm
  - no knowledge of scene’s geometry is required
  - must deal with aliasing artifacts

- Well known software rendering technique
  - Basic shadowing technique for Toy Story, etc.
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

- Compare with and without shadows
Visualizing Shadow Mapping

- The scene from the light’s point-of-view

FYI: from the eye’s point-of-view again
Visualizing Shadow Mapping

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

FYI: from the light’s point-of-view again
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

\textit{Green is where the light planar distance and the light depth map are approximately equal.} 

\textit{Non-green is where shadows should be.}
Visualizing Shadow Mapping

- Scene with shadows

Notice how specular highlights never appear in shadows

Notice how curved surfaces cast shadows on each other
Hardware Shadow Map Filtering

“Percentage Closer” filtering

- Normal texture filtering just averages color components
- Averaging depth values does NOT work
- Solution [Reeves, SIGGRAPH 87]
  - Hardware performs comparison for each sample
  - Then, averages results of comparisons
- Provides anti-aliasing at shadow map edges
  - Not soft shadows in the umbra/penumbra sense
Hardware Shadow Map Filtering

GL_NEAREST: blocky

GL_LINEAR: antialiased edges

Low shadow map resolution used to heighten filtering artifacts
Problems with shadow maps

- Hard shadows (point lights only)
- Quality depends on shadow map resolution (general problem with image-based techniques)
- Involves equality comparison of floating point depth values means issues of scale, bias, tolerance
Reflection Maps

Blinn and Newell, 1976
Environment Maps

Miller and Hoffman, 1984
Environment Maps

*Interface*, Chou and Williams (ca. 1985)
Environment Maps

180 degree fisheye
Photo by R. Packo

Cubical Environment Map

Cylindrical Panoramas
Reflectance Maps

- Reflectance Maps (Index by N)
- Horn, 1977
- Irradiance (N) and Phong (R) Reflection Maps
- Miller and Hoffman, 1984

Mirror Sphere  Chrome Sphere  Matte Sphere
Irradiance Environment Maps

Incident Radiance (Illumination Environment Map)

Irradiance Environment Map
Assumptions

- Diffuse surfaces
- Distant illumination
- No shadowing, interreflection

Hence, Irradiance a function of surface normal
Diffuse Reflection

\[ B = \rho E \]

Radiosity (image intensity) \rightarrow \downarrow \rightarrow \text{Reflectance (albedo/texture)} \rightarrow \text{Irradiance (incoming light)}

= \times \quad \text{quake light map}
Analytic Irradiance Formula

Lambertian surface acts like low-pass filter

\[ E_{lm} = A_l L_{lm} \]

\[ A_l = 2\pi \frac{(-1)^{l-1}}{(l+2)(l-1)} \left[ \frac{l!}{2^l \left( \frac{l}{2} \right)!} \right] \quad l \text{ even} \]

Ramamoorthi and Hanrahan 01
Basri and Jacobs 01
9 Parameter Approximation

Exact image

RMS error = 25 %

Order 0
1 term

$Y_{lm}(\theta, \varphi)$

$0 \leq l \leq m$
9 Parameter Approximation

Exact image

RMS Error = 8%

Order 1
4 terms

$Y_{lm}(\theta, \varphi)$

$l$
$m$

0
1
2

$xy$
$yz$
$3z^2 - 1$
$zx$
$x^2 - y^2$
9 Parameter Approximation

For any illumination, average error < 3% [Basri Jacobs 01]
Real-Time Rendering

\[ E(n) = n^t M n \]

Simple procedural rendering method (no textures)
- Requires only matrix-vector multiply and dot-product
- In software or NVIDIA vertex programming hardware

Widely used in Games (AMPED for Microsoft Xbox), Movies (Pixar, Framestore CFC, …)

```c
surface float1 irradmat (matrix4 M, float3 v) {
    float4 n = {v, 1};
    return dot(n, M*n);
}
```
Environment Map Summary

- Very popular for interactive rendering
- Extensions handle complex materials
- Shadows with precomputed transfer
- But cannot directly combine with shadow maps
- Limited to distant lighting assumption
Resources

- OpenGL red book (includes GLSL)
- Older books: OpenGL Shading Language book (Rost), The Cg Tutorial, ...
- [http://www.realtimerendering.com](http://www.realtimerendering.com)
  - Real-Time Rendering by Moller and Haines
  - Links to Miller and Hoffman original, Haeberli/Segal
- [http://www.cs.ucsd.edu/~ravir/papers/envmap](http://www.cs.ucsd.edu/~ravir/papers/envmap)
  - Also papers by Heidrich, Cabral, ...
- Lots of information available on web...