GDC: 06
WHAT'S NEXT
www.gdconf.com
GAME DEVELOPERSChoice Awards
INDEPENDENT GAMES FESTIVAL
GDC MOBILE
SERIOUS GAMES SUMMIT
GAME CONNECTION
March 20-24
San Jose, California
Game Developers Conference
Advanced Rendering Techniques with OpenGL ES 1.1+

Presented by Dan Ginsburg, Staff Engineer
Agenda

- Imageon 238x Rooms Demo Overview
- Advanced Rendering Techniques:
  - Lighting
  - Water
  - Postprocessing
  - Skinning
  - Particles
  - Reflections
Imageon 238x Rooms Demo
Imageon 238x Rooms Demo

- Designed to demonstrate ES 1.1+
- Uses nearly all features of API
- Attempts to minimize CPU calculations
- Techniques for your games
Lighting

- DOT3 Bumpmapping
- Projective Spotlight
- Per-pixel Cubemap Specular
- Shadows
Lighting Stages

- DOT3 Bump
- Spotlight
- Specular Cubemap
- Basemap
- Shadows
- Final
Stage 1 – DOT3 Bump
DOT3 - Challenges in ES 1.1+

- Goal: minimize CPU calculations
- Tangent-space bumpmapping:
  - Requires per-vertex matrix multiply on CPU
  - Requires dynamic vertex data
  - Far from ideal for handheld
DOT3 - Tangent vs. World Space

- **Tangent-space**
  - Matrix multiply per-vertex on CPU
  - Dynamic vertex data
  - Object can animate
  - Potentially less texture data

- **World-space**
  - No per-vertex CPU calculations
  - Larger texture data
  - Object is static
DOT3 in Lighting Demo

- Tangent-space on low-poly surfaces
- World-space on high-poly surfaces
  - Less CPU calculation
Stage 2 – Projective Spotlight
Projective Spotlight

- Use 2D spotlight texture image

- Projective texturing for spotlight effect
Projective Spotlight

- Send \((x, y, z)\) position as texcoord
- Compute texture matrix:
  - \(M = \text{Bias} \times \text{LP} \times \text{LV}\)
  - \(\text{LV} = \text{Modelview matrix from light}\)
  - \(\text{LP} = \text{Projection matrix from light}\)
  - \(\text{Bias} = \text{Bias matrix} \ [[-1, 1] \rightarrow [0, 1]]\)
Projective Spotlight Results

- Back projection from negative coordinates
Resolving Back Projection

- Need to fetch:
  - Black behind light near plane
  - White in front of light near plane

- Solution:
  - Use texture matrix to compute distance to plane
  - Use distance as texture coordinate
  - Texture has black at texel (0,0), white at (1,0)
Back projection texture matrix

- Put plane equation in texture matrix

\[
\begin{bmatrix}
P.x & 0 & 0 & 0 \\
P.y & 0 & 0 & 0 \\
P.z & 0 & 0 & 0 \\
P.w & 0 & 0 & 1 \\
\end{bmatrix}
\]

- Position \((x,y,z)\) in texture coordinate
- Combine with spotlight texture
Final Projective Spotlight

Without Back Projection Fix

With Back Projection Fix
Stage 3 – Specular Cubemap
Specular Highlight Cubemap
Applying Specular Cubemap

- Use REFLECTION_MAP texgen
- Generate texture matrix
  - Based on light direction
  - Use to rotate reflection vector
- Apply using texture combine
  - Optionally, attenuate by gloss channel
Stage 4 – Shadows
Stencil Technique

- ES 1.1+ enables true stencil shadows
- However, we used simple technique:
  - Compute “squash” matrix based on shadow-plane
  - Squash geometry using matrix
  - Use stencil buffer to mask shadow region
  - Also use stencil to write each pixel only once
Water

- Cubemap Fresnel Reflection
- Refraction
- Planar Reflections w/ Framebuffer Objects
- Lightmaps
Stage 1 – Refraction/Reflection
Refraction Mapping

- Water simulation on lattice of vertices
  - We used simple sin/cos waves
  - Algorithm is independent of simulation technique
- Dynamically compute refraction on CPU
  - Generate normal for each vertex in lattice
  - Use eye-vector and normal to generate refraction texcoords
- See: Game Programming Gems, *Refraction Mapping for Liquids in Containers*
Refraction Texture

- 2D view of closed container:

- Mapped using dynamic refraction coords
Reflection Mapping

- Prerender reflection cubemap

- FOV: 90 degrees.  Aspect: 1.0
Reflection Mapping

- REFLECTION_MAP texcoord generation
  - Generates eye-space reflection vector
  - …but we need reflection vector in world-space!

- Rotate reflection vector into world-space
  - Load inverse upper 3x3 of MV matrix
  - This is required to use REFLECTION_MAP with a moving camera!
Water Details

2D Refraction + (Cubemap Reflection * Fresnel)
Stage 2 – Planar Reflections
Dynamic Planar Reflection

- Create FBO for render-to-texture
- Flip camera about the ground plane
  - Also invert cull direction
- Draw scene to reflection texture
Rendering Reflections

- Use \((x, y, z)\) position as texture coordinate
- Compute texture-matrix to transform into projective-view space
  - Just like projective spotlight, except use camera matrix instead of light matrix
Puddles in a single pass

- Bind two textures:
  - Tex0 – RGBA basemap
    - RGB = base color
    - A = puddle opacity
  - Tex1 – Reflection texture (FBO)
- Use texture environment to combine in one pass
Postprocessing

- Light Bloom with Framebuffer Objects
- Diffuse Vertex Lighting
- Lightmaps
Light Bloom

128x128 64x64 32x32 16x16

10% 15% 34% 60%

Game Developers Conference
Postprocessing

- Render scene to framebuffer object
  - Clear screen to white
  - Draw all objects black
- Downsample to multiple sizes
- Blend all FBOs together
  - Use multitexture and alpha blend
- Blend combined FBO with final scene
Skinning

Matrix Palette
Skinning:
20 Bones
4 Weights/Bone
5766 Polys

Per-pixel Cubemap Specular

Gloss Specular Attenuation
Lighting the Skinned Model

- Use GL lighting for vertex diffuse
- Use specular cubemap for per-pixel specular
  - Same as in lighting demo
- Combine together in a single pass
Lighting – Texture Environment

- Tex0
  - RGB = Base color
  - A = Gloss

- Tex1
  - RGB = <unsed>
  - A = Specular

- Texture environment:
  - VertexColor.rgb * Base.rgb + Specular.a * Base.a
Skinning – Lighting in a Single Pass

- Basemap
- Vertex Light
- Specular * Gloss
Particles

Distance Attenuated Point Sprites

Flicker Attenuated Lightmaps
Particle System for ES

- Generic particle emitters
- Modify properties over particle lifetime:
  - Position
  - Color
  - Size
- ES allows per-particle size
  - Desktop OpenGL did not easily allow this
Distance Attenuation

- Point sprites have some limitations
  - Texture coordinates are fixed [0,0] -> [1,1]
  - Size is expressed in screen pixels
- Point parameters must be used
  - Attenuate particles by distance to camera
  - Account for resolution change
Point Parameters

- Input \((a, b, c)\) into attenuation equation
  
  \[
  \text{atten}(d) = \frac{1}{a + b \cdot d + c \cdot d^2}
  \]

- We compute scale based on resolution
  - \(a = (\text{scale} \cdot \text{scale})\)
  - \(b = 0\)
  - \(c = \text{scale} \cdot 0.15\)

- This gives proper scaling by resolution change and distance to eye
Reflect

- Dynamic Cubemap with Framebuffer Objects
- Blended Shadows
- Dynamic Diffuse Vertex Lighting
Dynamic Cubemap

- Create 6 FBOs, one for each cube face
- Each frame:
  - Render scene to each cube face
  - Use proxy geometry for performance
  - Also use frustum culling
Applying the Cubemap

- As in Water demo:
  - Use REFLECTION_MAP on globe
  - Use texture matrix to rotate from eye-space to world-space
- Blend with basemap and vertex lighting
Summary

- ES 1.1+ enables advanced techniques
  - Lighting
  - Water
  - Postprocessing
  - Skinning
  - Particles
  - Reflections
Conclusion

● Differentiate your game with advanced features

● Think a bit differently…
  • …do as little as possible on the CPU

● Get creative…
  • These are just some examples, ES 1.1+ enables much more!
More info…

- Khronos: [www.khronos.org](http://www.khronos.org)
  - Imageon 238x Rooms Demo
    - Whitepaper
    - Movies
    - Screenshots
Questions?