Image-based HDR Lighting as Examples

Programmable Rendering Pipeline

王銓彰
墨匠科技 BlackSmith CEO
cwang001@mac.com
kevin.cwang3@mac.com
cwang001@blacksmis.com
3D Rendering Pipeline

• Before GPU era (Before 2005)
  • Fixed-function rendering pipeline
    • Wireframe (1980)
    • Gouraud Shading (Lighting on Vertex) (1990)
      • Using Phong Reflection Model
    • Texture mapping (NVidia nv-1, 1995)
    • Multi-layered texture mapping (2000)
3D Rendering Pipeline

- When GPU is coming
  - Computation is cheap and fast !!!
  - Floating-point mathematics in hardware
  - Vectorized acceleration
    - Single introduction with multiple data (SIMD)
  - Parallel in vertex/pixel threading
    - Thousands of Cores
  - $\sim 1000x$ faster than CPU
- A tiny personal super computer
3D Rendering Pipeline

- Now in programmable rendering pipeline
  - Vertex processing
    - Vertex Shader
  - Pixel processing (partial, fragment processing)
    - Pixel Shader (Fragment Shader)
  - Procedural processing in geometry
    - Geometry shader
  - Tessellation by GPU (Hull shader + Domain Shader)
  - GPGPU (General purpose GPU)
    - Compute Shader
    - NVIDIA CUDA
Shader

- A small program (function) embedded in rendering pipeline and invoked by graphic command and running on GPU
- Basically it’s assembly but now popular in high-level language form.
  - C-like graphics language
  - Native in vector mathematics
    - 128-bit registers
- Popular solutions
  - NVIDIA Cg
  - MS HLSL (High-level Shading Language)
  - OpenGL GLSL
High-quality Real-time Rendering Effects

- Cook & Torrance Material
- Reflection
- Sub-surface Scattering
- Parallax Occlusion Map
- High Dynamic Range
- Image-based Lighting
Vertex Processing

- At least three 4x4 transformation matrices in this processing
  - World matrix
    - From local space to world space
  - View matrix
    - From world space to view space
  - Projection matrix
    - Orthogonal or perspective
    - From view space to projection space

\[ v' = v \cdot M_{\text{world}} \cdot M_{\text{view}} \cdot M_{\text{Projection}} \]
Vertex Processing

- Skin deformation is always calculated in world space
  - Vertex blending
- After viewing, vertex fog is added as necessary (Suggested)
- Lighting is calculated on vertices (Suggested)
  - Same as in fixed function 3D rendering pipeline
- After the vertex processing:
  - The vertices are in projection space.
  - Vertex position: (x, y, z, w)
  - w > 1.0 (input = 1.0)
- Vertex processing is programmable.
  - Vertex shader
Primitive Processing

- After homogeneous divide,
  - $-1 < x < 1$
  - $-1 < y < 1$
  - $0 < z < 1$
- Then mapping:
  - Scale $(x, y)$ to viewport
  - Map $z$ value to $z$ buffer
- Triangle setup
  - Setup the triangle edge
  - Bi-linear interpolation using vertex data
- Converting per-vertex attributes to per-pixel attributes
- Non-programmable
Pixel Processing

- Pixel processing has the most computing cost in the rendering pipeline.
- Two stages:
  - Stage I: fragment processing
    - Programmable
  - Stage II:
    - Non-programmable
- In stage I:
  - GPU samples multiple texture data according to the interpolated texture coordinates.
  - Then blend per-pixel attributes:
    - Diffuse and specular colors
    - Multiple texture samples
Pixel Processing

- The lited/textured pixel data in \((r, g, b, a)\) is generated after the fragment processing.
- In stage II:
  - Several tests are performed if the pixel affects the final pixel color:
    - Alpha test
    - Depth test
    - Z buffer test
    - Stencil test
  - In the alpha blend:
    - Apply pixel alpha to create a semi-transparent blend between a source pixel and frame buffer pixel.
- Only the pixel processing stage I is programmable.
Single-pass Rendering

One Rendering Pass
**Multi-pass Rendering**

- Shader is a short/single-function program with highly parallelism
- Textures are saving in video memory and acting as a memory block
- We can render into textures (rendering targets)
Multi-pass Rendering

- Program is divided into (shaders) and use textures as memory blocks.
- Use Draw command to trigger the shader to execute
- Vertex shader and pixel shader are gathering data from textures
- Not broadcasting data to neighboring vertices and pixels
HDR As Post-processing

- Scene
- HDR Texture Image
- Bloom Filter
- +
- Tone Map
- LDR Screen
Real-time HDR – Step I

- Render the 3D scene on a HDR texture
  - RGB format on screen is LDR:
    - R8G8B8A8
    - (0, 0, 0) ~ (255, 255, 255)
  - HDR:
    - RGBA > 16 bits
    - floating-point real number
    - color value > 1.0
Real-time HDR – Step II

- Get the bright part & bloom it
Real-time HDR – Step II

• Another way to get the bright pass:
  • Calculate the luminance of the image pixel
  • Multiple the pixel with luminance: (Shader code)

```c
float luminance = dot(color, float3(0.299f, 0.587f, 0.114f));
float4 result = color * luminance;
```
Real-time HDR - Step II

- Blooming
  - Using Kawase bloom filter in 8 passes, any more passes result in more blur.

![Diagram](image-url)
Real-time HDR – Step III

- Tone mapping
  - Mapping HDR Image to LDR of screen
Real-time HDR – Step III

• Reinhard Tone Mapping
• Basic idea:
  • LDR = HDR / (1 + HDR)
• Formula:

\[
\text{Color}(x,y) = \frac{\text{FullScreenImage}_\text{Scaled}(x,y) \cdot (1 + \text{FullScreenImage}_\text{Scaled}(x,y) / \text{Lum}_{\text{White}}^2)}{1 + \text{FullScreenImage}_\text{Scaled}(x,y)}
\]
HDR Post-processing

2nd pass: render the scene to a render target texture

3rd pass: scale & bright pass

4th-11th pass: bloom

12th pass: tone mapping

No HDR

HDR
HDR Post-processing Demo
HDR for Image-based Lighting

- Image-based lighting
- Hollywood has been doing this since the 90’s
- Now possible in real time
HDR for Image-based Lighting

- Using HDR images as light sources
  - IBL in short
- Without IBL
  - Harsh
  - Simplistic
  - Noticeably computer generated
HDR for Image-based Lighting

- With IBL
  - Increasing scene’s level of realism
  - Increasing the visual interest
- Key technologies
  - HDR photography
  - Omnidirectional photography
  - Global illumination techniques
Image-based Lighting Example
Real-time Image-based Lighting

- Diffuse environment lighting (Shader programming)
  - Diffuse environment map
  - Specular environment map
- Take the 360-degree panorama HDR picture as the environment map.
  - Use HDRShop to generate the diffuse/specular environment map
Real-time Image-based Lighting

Sampling on semi-sphere

$$\text{sumColor} = \sum \text{color}[i] \cdot \text{dot}(\mathbf{N}, L_i)^n$$

\[ n = 1 : \text{baking diffuse environment map} \]
\[ \text{otherwise} : \text{shininess parameter in Phong Reflection Model} \]
Real-time Image-based Lighting

float2 LatLongIM(float3 v)
{
    float3 vv = normalize(v);
    float theta = acos(vv.z);  // we use +z up
    float phi = atan2(vv.x, vv.y) + 3.1415962;

    return float2(phi, theta)*float2(0.15916, 0.31831);
}

...  
float2 uv = LatLongIM(normDir);
imgDiff = probeDMap.Sample(probeDMapSampler, uv)*probeScaleD;

float3 refl = reflect(-camDir, normDir);
float2 uv = LatLongIM(refl);
imgSpec = probeSMap.Sample(probeSMapSampler, uv)*probeScaleS;

// image-based lighting
float4 rgba = imgDiff*diffuseMat + imgSpec*specularMat;
Image-based Lighting Demo