3D Authoring Tool “BS Content Studio” supports Deferred Rendering for improved visual quality

Oliver Neubauer
Project Manager

02.07.2013

www.bitmanagement.com
BS Content Studio

• BS Content Studio manages hundreds of lights
• WYSIWYG editor to create, manipulate and enrich your 3D models – authoring tool facilitates content generation
• visualisation in integrated 3D engine “BS Contact”
• Deferred Rendering reduces complexity of 3D scene

www.bitmanagement.com
Concept

• Improve the lighting
• Get rid of the light limitation (8 HW lights)
• More light → more realism
• Decouple the lighting of object from the rendering of object
History

• Inventor 1988 Michael Deering et al.
  – Pixel colour calculation after resolving depth

• Current concept from 1990 Saito and Takahashi
  – Introduce the G-Buffer
Forward rendering

• Classical forward rendering for each pixel per object
  – Determine depth (culled or not)
  – Normals + diffuse color + light color = final color
• Each pixel has to be rendered for every light
• Complexity $O(m*n)$
  – $m$ number of object
  – $n$ number of lights
Forward rendering

- Shading is done in place
- HW lighting depends on vertex density
Deferred Lighting

• 1. Pass collects geometry information
• G-Buffer (Geometry Buffer) contains information
  – Depth
  – Diffuse colour
  – Normal
• G-Buffer are MRT (multiple render targets) textures
Deferred Lighting

DEPTH

COLOUR

NORMALS
Deferred Lighting

• 2. Pass collects Light information

Directional light
Deferred Lighting

3. Pass combine light and geometry information
Transparency

• Transparency is complicated
• Transparent object receive light and blends with scene
• Transparent object use the old forward rendering style
Transparency

• Scene with transparent object
• Light on transparent object use forward shading
• Transparent blends with deferred rendered object
• Pros

- multiple lights for objects
- complexity $O(m+n)$ $m = \text{Object}$; $n = \text{lights}$
- No limits of hardware lighting
- Only visible geometry get lighted
- Shadow maps easier to maintain
- Post effects easy to add
Pro & Contra

• Contra

  – Transparent objects hard to handle
  – Driver and graphic cards need MRT support
  – Shader Model 3 required
    • DX9
    • OGL version 2
  – Currently not available for OGLES 2.0
“BS Contact” 3D Engine

• New node **DeferredNode** in X3D Syntax

```xml
<DeferredNode>
  <PackagedShader containerField='globalShader' /> <!-- MRT Writer -->
  <PackagedShader containerField='lightShader' /> <!-- light shader -->

  <CompositeTexture3D containerField='renderTargets' parameter="mipmap=false" "format=R32F" "depth=D24X8"/>
  <!-- MRT for depth -->
  <CompositeTexture3D containerField='renderTargets' parameter="mipmap=false" "format=A8R8G8B8" "depth=NONE"/>
  <!-- MRT diffuse -->
  <CompositeTexture3D containerField='renderTargets' parameter="mipmap=false" "format=A8R8G8B8" "depth=NONE"/>
  <!-- MRT normal -->

  <CompositeTexture3D containerField='lightRenderTarget' parameter="mipmap=false" "format=A8R8G8B8" "depth=NONE"/>
  <!-- MRT light color -->

  <PackagedShader containerField='combinePostProcess' />
  <!-- combine shader -->

</DeferredNode>
```
Children contains nodes for deferred lighting

MRT must have same bit rate
- A8R8G8B8
- R32F
- G16R16

GlobalShader field for MRT shader writer

MRT's are filled in one pass
DX9 HLSL pixel shader example for material MRT

PO PS_Colors_material(in VO input)
{
    PO result = (PO)0;
    result.normal = 0.5f * (normalize(input.normal) + 1.0f);
    result.depth = input.depth.x / input.depth.y;
    result.normal.a = material.power/128;

    result.diffuse.rgb = material.diffuseColor.rgb*input.color.rgb;
    result.diffuse.a = 1;
    return result;
}
1. RT is depth with 32 bit precision
2. RT is colour 8bit for each RGB channel
   - Last 8 bit are free to use (emissive colour factor?)
3. RT is normal 8bit precision for each axis xyz
   - Lead to quantization
   - Solution 16bit for x and y axis reconstruct z axis
• LightShader field for light colour calculation
• Light is rendered as geometry
• For each light type separate shader
• Result Shader information stored in render target
float4 PixelShaderDirectionalFunction(VertexShaderOutput input) : COLOR0
{
    // get normal data from the normalMap
    half4 normalData = tex2D(normalSampler,input.TexCoord);

    // transform normal back into [-1,1] range
    half3 normal = 2.0f * normalData.xyz - 1.0f;

    // get specular power, and get it into [0,255] range
    half specularPower = normalData.a*128;

    // read depth
    float depthVal = tex2D(depthSampler,input.TexCoord).r;

    // compute screen-space position
    float4 position;
    position.x = input.TexCoord.x * 2.0f - 1.0f;
    position.y = -(input.TexCoord.y * 2.0f - 1.0f);
    position.z = depthVal;
    position.w = 1.0f;

    // transform to world space
    position = mul(position, g_mViewProjInvers);
    position /= position.w;
}
//surface-to-light vector
float3 lightVector = -normalize(light.direction);

//compute diffuse light
half NdL = max(0, dot(normal, lightVector));
half3 diffuseLight = NdL * light.diffuseColor;

//reflection vector
half3 reflectionVector = (reflect(lightVector, normal));

//camera-to-surface vector
half3 directionToCamera = normalize(cameraPos - position);

//compute specular light
half dotProd = dot(reflectionVector, directionToCamera);

half specularLight = 0;
if(specularPower > 0 && NdL > 0)
    specularLight = /*specularIntensity */ pow(saturate(dot(reflectionVector, -directionToCamera)), specularPower);

//output the two light values
return float4(diffuseLight.rgb, max(0, specularLight)) ;
BS Contact

• LightRenderTarget field contains result from shader
• 32 bit texture RGB channels contains light colour
• 8 bit Alpha channel for specularity
Light RT
with
98 Lights
and
random
colour

All lights
BS Contact

• combinePostProcess field for PostProcess node to process results from colour RT and light RT
• PostProcess node contains shader for combine process
• Chaining of PostProcess nodes are flexible to add own effects
• Post process effects simple to implement using the already computed RT
  – SSAO
  – Shadow
  – Blur
  – Bloom
  – Motion Blur
BS Contact

SSAO Buffer
BS Contact

No SSAO

SSAO
BS Contact

• BS Contact can handle hundreds of lights from different types
• Performance depends on size and range of light
• Directional light is costly because full scene lighting
• Small point lights could be cheap
BS Contact

• 100 point lights
• 98 spot lights
• all animated
Deferred Rendering effects simply applied (placing of lights in interactive 3D scene):

BS Content Studio               Result
• Demo Scene available on
http://www.bitmanagement.de/en/company/research-development