CG Programming: 3D Texturing

- 3D texturing is used as image based rendering
- Applications
  - 3D texture mapping for complicate geometric objects
    * It generates highly natural visual effects in which objects appear carved from lumps of materials rather than laminated with thin sheets as in 2D texture mapping.
  - Volume rendering
    * Ray casting
    * 3D texture mapping-based rendering
3D Texture Mapping

Subsection 2.1 3D Texture Modeling

Polyagonal Model with 3D Texture

Subsection 2.2 3D Texture Cell Selection

Polyagonal Model with Selected 3D Texture Cells

Subsection 2.3 3D Texture Compression

Polyagonal Model with Compressed 3D Texture

Subsection 2.4 Polygonal Rendering

Final Raster Image
3D Texture Mapping Results

(a) Teapot with Bmarble

(b) Dragon with Wood

(c) Bunny with Eroded

(d) Dragon with Wood
Volume Rendering and CG Programs

Volume rendering describes techniques that allow the visualization of three-dimensional image data.

- Ray casting
- 3D Texture mapping-based rendering
- Iso-surface generation (marching cubes)
3D Volumetric Images (CT scan)
Volume Rendering Results
At each volume sampling point \((x, y, z)\) along the ray, function values are estimated using tri-linear interpolation. Furthermore illumination \(I(x, y, z)\) reaches each sampling point from the light source \(L\).

\[ C(t), \alpha(t) \]

\( C(t) \): a shaded color computed by an illumination model
\( \alpha(t) \): an opacity value set by Transfer function maps (more about this below)
Illumination

\[ I = K_a I_a + K_d (\vec{N} \cdot \vec{L}) + K_s (\vec{N} \cdot \vec{H})^n \]

\( \vec{L} \): Light vector

\( \vec{V} \): View vector

\( \vec{H} = \text{Normalize}(\vec{L} + \vec{V}) \)

Gradient at \( x_i \):

\[ \nabla f(x_i) = \]

\[ \left( (f(x_{i+1}, y_i, z_i) - f(x_{i-1}, y_i, z_i))/D_x, \right. \]

\[ \left( f(x_i, y_{i+1}, z_i) - f(x_i, y_{i-1}, z_i)/D_y, \right. \]

\[ \left( f(x_i, y_i, z_{i+1}) - f(x_i, y_i, z_{i-1})/D_z) \right) \]

\[ \vec{N}_i = \|\nabla f(x_i)\| \]
Ray Casting - revisited

- Advantages
  - Not necessary to explicitly extract surfaces from volume when rendering
  - Can change the transfer functions to make various surfaces stand out within the volume

- Disadvantages
  - Do not have explicit representations for surfaces, therefore not straightforward to compute integral/differential properties
  - Much more computationally intensive to render volume since not dealing directly with the efficient polygon pipeline
• Front-to-back composition
  - $C_d = \alpha_s C_s + (1 - \alpha_s) C_d$
  - $\alpha_d = \alpha_s + (1 - \alpha_s) \alpha_d$

• Back-to-front composition
  - $C_d = C_d + (1 - \alpha_d) \alpha_s C_s$
  - $\alpha_d = \alpha_d + (1 - \alpha_d) \alpha_s$

• Color and Opacity Symbols
  - $C_d$ : Destination color
  - $C_s$ : Source color
  - $\alpha_d$ : Destination opacity
  - $\alpha_s$ : Source opacity
Transfer Functions

- Transfer functions make a volume dataset visible by assigning renderable properties such as opacities (and color)
- 1D transfer functions:
  - Intensity $\rightarrow$ RGB$\alpha$
- 2D transfer functions:
  - Intensity, Gradient Magnitude $\rightarrow$ RGB$\alpha$

For details, see: S. Park, C. Bajaj, "Feature Selection of 3D Volume Data through Multi-Dimensional Transfer Functions" Pattern Recognition Letters, 28, 3, pp. 367-374, 2007
The Contour Spectrum

- The contour spectrum allows the selection of transfer functions

3D Texture based Volume Rendering

Ray Casting

3D Texture Mapping

View point aligned slicing
Shading in Graphics Hardware

Light Vector (L)

RGB Normal Vectors (N)
3D Texture

Half Vector (H)

Volume Data (I)
3D Texture

Dot Product
(L\cdot N)

Dot Product
(H\cdot N)

Diffuse
(L\cdot N)

Specular
(H\cdot N)

Power

Specular
(H\cdot N)^n

RGB Color*Diffuse + Specular

RGB Color

Opacity(\alpha)

Register Combiners

Color Lookup Table
Dependent Texture

Opacity(\alpha)

RGB

The University of Texas at Austin
Shading in Graphics Hardware (contd)

3D indexed volume data → Color Look-up Table

Normal vectors as RGB texture → Normal map

Texture mapped color images → Dot products with a light vector for diffuse

Pixels are combined by the register combiners → Shaded final image

- Index | R | G | B | A |
- 0 | 0 | 0 | 0 | 0 |
- 1 | 10 | 10 | 0 | 100 |
- ... |
- 255 | 100 | 0 | 255 | 255 |
Register Combiners

Nvidia Register Combiners

Per-fragment operations (Programmable)
Cg Language

- C for graphics (or C for GPUs)
- High-level shading language for programming GPUs
- Developed by NVIDIA
- Works for both NVIDIA and ATI GPUs
Cg Vertex Program for Volume Rendering

void main( float4 Position : POSITION,
           float3 TexCoor : TEXCOORD0,
           uniform float4x4 ModelViewProj,
           out float4 oPosition :POSITION,
           out float3 oTexCoor :TEXCOORD0)
{
    oPosition = mul(ModelViewProj, Position);
    oTexCoor = TexCoor;
}
Cg Fragment Program for Volume Rendering

```cpp
float3 Expand(float3 Vector) { return (Vector - 0.5)*2.0; } 

void main ( float3 TexCoor : TEXCOORD0, // Texture Coordinates 
            out float4 oColor : COLOR, // Output RGBA
            uniform sampler3D RGB_NormalMap, // Voxel Normal
            uniform sampler3D Data_I, // Data Index
            uniform sampler1D TF, // User Defined TF
            uniform float3 Lightv, // Light Vector
            uniform float3 Halfv) { // Half Vector
    float3 Normal = Expand(tex3D(RGB_NormalMap, TexCoor).rgb);
    float Diffuse = abs(dot(Normal, Lightv)); // Diffuse Color
    float Specular = abs(dot(Normal, Halfv)); // Specular Color
    float Power8 = pow(Specular, 8); // 8th Power of Specular
    float3 Specularf3 = float3(Power8, Power8, Power8);
    float Intensity = tex3D(Data_I, TexCoor); // (I) of Each Voxel
    float4 RGBA = tex1D(TF, Intensity);
    oColor.w = RGBA.w;
    oColor.xyz = (Diffuse*RGBA.xyz + Specularf3)*oColor.w; }
```
Cg Programs with OpenGL

Step 1: Cg profile selection

CGcontext contextCg_g;
CGprogram vertexProgramCg_g, fragmentProgramCg_g;
CGprofile vertexProfileCg_g, fragmentProfileCg_g;

contextCg_g = cgCreateContext();

// Vertex profile
if (cgGLIsProfileSupported(CG_PROFILE_VP30))
    vertexProfileCg_g = CG_PROFILE_VP30;
else {
    if (cgGLIsProfileSupported(CG_PROFILE_ARBVP1))
        vertexProfileCg_g = CG_PROFILE_ARBVP1;
    else exit(1);
}
// Fragment profile
if (cgGLIsProfileSupported(CG_PROFILE_FP30))
    fragmentProfileCg_g = CG_PROFILE_FP30;
else {
    if (cgGLIsProfileSupported(CG_PROFILE_ARBFP1))
        fragmentProfileCg_g = CG_PROFILE_ARBFP1;
    else exit(1);
}

Cg Programs with OpenGL

Step 2: Cg program loading and compile

vertexProgramCg_g = cgCreateProgramFromFile(
                    contextCg_g, CG_SOURCE, "vertex.cg",
                    vertexProfileCg_g, NULL, NULL);
if (!cgIsProgramCompiled(vertexProgramCg_g))
   cgCompileProgram(vertexProgramCg_g);
cgGLEnableProfile(vertexProfileCg_g);
cgGLLoadProgram(vertexProgramCg_g);

fragmentProgramCg_g = cgCreateProgramFromFile(
                       contextCg_g, CG_SOURCE, "fragment_FTB.cg",
                       fragmentProfileCg_g, NULL, NULL);
if (!cgIsProgramCompiled(fragmentProgramCg_g))
   cgCompileProgram(fragmentProgramCg_g);
cgGLEnableProfile(fragmentProfileCg_g);
cgGLLoadProgram(fragmentProgramCg_g);
Cg Programs with OpenGL

Step 3: Texture Loading

```c
glTexImage3D(GL_TEXTURE_3D, 0, GL_COLOR_INDEX8_EXT, W, H, D, 0,
             GL_COLOR_INDEX, GL_UNSIGNED_BYTE, read_datauc);
cgGLSetTextureParameter(
    cgGetNamedParameter(fragmentProgramCg_g, "Data_I"),
    color_index_3DT.getTextureID());

glTexImage3D(GL_TEXTURE_3D, 0, GL_RGBA, W, H, D, 0,
             GL_RGBA, GL_UNSIGNED_BYTE, RGBNormals);
cgGLSetTextureParameter(
    cgGetNamedParameter(fragmentProgramCg_g, "RGB_NormalMap"),
    RGB_normals_3DT.getTextureID());
```
glTexImage1D(GL_TEXTURE_2D, 0, GL_RGBA, 256,
        0, GL_RGBA, GL_UNSIGNED_BYTE, TF_guc);

cgGLSetTextureParameter(
        cgGetNamedParameter(fragmentProgramCg_g, "TF"),
        TF_1DT.getTextureID()));
Cg Programs with OpenGL

Step 4: Parameter set-up

CGparameter param;

cgLSetStateMatrixParameter(
    cgGetNamedParameter(vertexProgramCg_g, "ModelViewProj"),
    CG_GL_MODELVIEW_PROJECTION_MATRIX, CG_GL_MATRIX_IDENTITY);

// Vertex Program
// Set Parameter Pointers for the Vertex Program
param = cgGetNamedParameter(vertexProgramCg_g, "Position");
cgLSetParameterPointer(param, 3, GL_FLOAT, 0, Vertices_f);
param = cgGetNamedParameter(vertexProgramCg_g, "TexCoor");
cgLSetParameterPointer(param, 3, GL_FLOAT, 0, TexCoor_f);
// Enable the variables
CGLEnableClientState(cgGetNamedParameter(vertexProgramCg_g, "Position"));
CGLEnableClientState(cgGetNamedParameter(vertexProgramCg_g, "TexCoor"));
// Fragment Program
// Set Parameter Pointers for the Fragment Program
param = cgGetNamedParameter(fragmentProgramCg_g, "Lightv");
cgGLSetParameter3fv(param, &lightV3f[0]);
param = cgGetNamedParameter(fragmentProgramCg_g, "Halfv");
cgGLSetParameter3fv(param, &h_viewV3f[0]);

// Enable the variables
cgGLEnableTextureParameter(
        cgGetNamedParameter(fragmentProgramCg_g, "RGB_NormalMap"));
cgGLEnableTextureParameter(
        cgGetNamedParameter(fragmentProgramCg_g, "ColorIndexMap"));
cgGLEnableTextureParameter(
        cgGetNamedParameter(fragmentProgramCg_g, "TF"));
Cg Programs with OpenGL

Step 5: Drawing Geometric Elements

// Drawing Triangles
glDrawElements(GL_TRIANGLES, 3*NumTriangles, GL_UNSIGNED_INT, Indices_i);

// Disable the variables
cgGLDisableClientState(cgGetNamedParameter(vertexProgramCg_g, "Position"));
cgGLDisableClientState(cgGetNamedParameter(vertexProgramCg_g, "TexCoor"));

// Disable the texture
cgGLDisableTextureParameter(
    cgGetNamedParameter(fragmentProgramCg_g, "RGB_NormalMap"));
cgGLDisableTextureParameter(
    cgGetNamedParameter(fragmentProgramCg_g, "ColorIndexMap"));
cgGLDisableTextureParameter(
    cgGetNamedParameter(fragmentProgramCg_g, "TF"));
More 3D Texture-Based Volume Rendering Results
Reading Assignment and News

Please review the appropriate sections related to this lecture in chapter 7, and associated exercises, of the recommended text.

(Recommended Text: Interactive Computer Graphics, by Edward Angel, Dave Shreiner, 6th edition, Addison-Wesley)

Please track Blackboard for the most recent Announcements and Project postings related to this course.

(http://www.cs.utexas.edu/users/bajaj/graphics2012/cs354/)