

Lab 5 GPU Implementation Instructions

(by Ziyi Zheng)

Here is the basic algorithm for the GPU rendering:

- Store the dataset as a 3D texture, where the bounding box is a box (of the size of the volume) with the texture coordinates at the eight corners encoded into the RGB color channels.
- Render the front faces of this cube and store the result on the FBO.
- Render the back faces into another area of the FBO and subtract the color values of the back face from the front face. Then, normalize the result and store it in a 2D texture. This will give you the direction vectors of the casting ray for each pixel. The ray origins are given by the value of the projected bounding box front faces.
- Cast a ray for each pixel into the 3D texture volume and sample the volume along the ray with the given step size (write a for-loop in your fragment program – for older GPUs the loop length is limited, you may have to break it up into a double loop)
- When the ray leaves the volume or the accumulated opacity exceeds the preset opacity threshold (typically 0.95), terminate the ray

FBO

1. Create FBO

```
void Application::CreateBuffer(GLuint fbo, GLuint *outputTexture,int num)
{
    glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
    glBindTexture(GL_TEXTURE_RECTANGLE_NV, outputTexture[num]);
    glTexImage2D(GL_TEXTURE_RECTANGLE_NV, 0, GL_RGBA32F_ARB,
mainwindow->pViewer->w(), mainwindow->pViewer->h(), 0,
                GL_RGBA, GL_FLOAT, NULL);

    // the following sets up linear interpolation, replace by GL_NEAREST for nearest
neighbor interpolation
    glTexParameterf(GL_TEXTURE_RECTANGLE_NV, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
    glTexParameterf(GL_TEXTURE_RECTANGLE_NV, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);

    // specifies what to do when exceeding the texture boundaries, here clamp to edge
    glTexParameterf(GL_TEXTURE_RECTANGLE_NV, GL_TEXTURE_WRAP_S,
GL_CLAMP_TO_EDGE);
    glTexParameterf(GL_TEXTURE_RECTANGLE_NV, GL_TEXTURE_WRAP_T,
GL_CLAMP_TO_EDGE);
```

```

        glFramebufferTexture2D(GL_FRAMEBUFFER_EXT,attachmentpoints[num],
            GL_TEXTURE_RECTANGLE_NV, outputTexture[num], 0);
    }

```

2. Error-Reporting function

```

void CheckFramebufferStatus()
{
    GLenum status;
    status = (GLenum) glCheckFramebufferStatusEXT(GL_FRAMEBUFFER_EXT);
    switch(status) {
case GL_FRAMEBUFFER_COMPLETE_EXT:
    break;
case GL_FRAMEBUFFER_UNSUPPORTED_EXT:
    printf("Unsupported framebuffer format\n");
    break;
case GL_FRAMEBUFFER_INCOMPLETE_MISSING_ATTACHMENT_EXT:
    printf("Framebuffer incomplete, missing attachment\n");
    break;
    // case GL_FRAMEBUFFER_INCOMPLETE_DUPLICATE_ATTACHMENT_EXT:
    //     printf("Framebuffer incomplete, duplicate attachment\n");
    //     break;
case GL_FRAMEBUFFER_INCOMPLETE_DIMENSIONS_EXT:
    printf("Framebuffer incomplete, attached images must have same dimensions\n");
    break;
case GL_FRAMEBUFFER_INCOMPLETE_FORMATS_EXT:
    printf("Framebuffer incomplete, attached images must have same format\n");
    break;
case GL_FRAMEBUFFER_INCOMPLETE_DRAW_BUFFER_EXT:
    printf("Framebuffer incomplete, missing draw buffer\n");
    break;
case GL_FRAMEBUFFER_INCOMPLETE_READ_BUFFER_EXT:
    printf("Framebuffer incomplete, missing read buffer\n");
    break;
default:
    printf("others!");
    return ;
    }
}

```

3. Bind FBO with depth-Buffer, depth buffer is needed in our entry-exit point algorithm.

```

glGenFramebuffersEXT(1, &fbo); //generate FBO
glGenTextures(2, outputTexture); //generate 2 textures
    CreateBuffer(fbo,outputTexture,0); //attach 2 textures with FBO
    CreateBuffer(fbo,outputTexture,1);

```

```

glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
glGenRenderbuffersEXT(1, &depthBuffer); //generate depth buffer
glBindRenderbufferEXT(GL_RENDERBUFFER_EXT, depthBuffer);
glRenderbufferStorageEXT(GL_RENDERBUFFER_EXT,
GL_DEPTH_COMPONENT24, mainwindow->pViewer->w(),
mainwindow->pViewer->h());
glFramebufferRenderbufferEXT(GL_FRAMEBUFFER_EXT,
GL_DEPTH_ATTACHMENT_EXT, GL_RENDERBUFFER_EXT, depthBuffer);
CheckFramebufferStatus()//check for errors
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);

```

4. Render in off-screen mode, into the FBO. After GPU rendering the entry-exit data is stored in 2 textures. They do not need to be displayed.

```

glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
//render something here
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);

```

ModelViewMatrix + TrackBall

We need the eye position to calculate the ray direction. We also need the light position to calculate the lighting effects (for lab 6). We are going to perform a series of coordinate transformations. First we perform a transform from global coordinates into model coordinates. Then we perform a transform from model coordinates into texture coordinates.

1. Obtain modelview matrix from opengl

```

GLfloat pmodelviewMatrix[16];
glGetFloatv(GL_MODELVIEW_MATRIX,pmodelviewMatrix);
glActiveTexture(GL_TEXTURE0);

```

2. Transform eye position from world coordinate into model coordinate.

```

//eye position
// set eye back (0,0,argument[0]) into world coordinate
double x= pmodelviewMatrix[2]*argument[0];
double y = pmodelviewMatrix[6]*argument[0];
double z = pmodelviewMatrix[10]*argument[0];

```

3. Calculate light position (do this for lab 6) in world coordinate. Light position is stored in track ball.

```

Tlight.convert(lmatrix);
//offset of light bulb 0.5*(vol_w,vol_h,vol_d) before rotation
float lx =
lmatrix[3][0]+(lmatrix[0][0]*vol_w+lmatrix[1][0]*vol_h+lmatrix[2][0]*vol_d)*0.5;
float ly =

```

```

lmatrix[3][1]+(lmatrix[0][1]*vol_w+lmatrix[1][1]*vol_h+lmatrix[2][1]*vol_d)*0.5;
float lz =
lmatrix[3][2]+(lmatrix[0][2]*vol_w+lmatrix[1][2]*vol_h+lmatrix[2][2]*vol_d)*0.5;

```

3. Transform light position from world coordinates into model coordinates. Light position is stored in the track ball.

```

(~T).convert(lmatrix);
// convert light (lx,ly,lz) into object orientation
float nlx = (lmatrix[0][0]*lx+lmatrix[1][0]*ly+lmatrix[2][0]*lz);
float nly = (lmatrix[0][1]*lx+lmatrix[1][1]*ly+lmatrix[2][1]*lz);
float nlz = (lmatrix[0][2]*lx+lmatrix[1][2]*ly+lmatrix[2][2]*lz);

```

4. Transform eye and light positions from model coordinates into texture coordinates. Finally pass these positions into the cg program.

```

//shift 0.5*(vol_w,vol_h,vol_d) and set into object scale (1/vol_w,1/vol_h,1/vol_d)
cgGLSetParameter3f(LightPositionr,(nlx+0.5*vol_w)/ vol_w, (nly+0.5*vol_h)/ vol_h,
(nlz+0.5*vol_d)/ vol_d);
cgGLSetParameter3f(EyePositionr,(x+0.5*vol_w) / vol_w, (y+0.5*vol_h)/ vol_h,
(z+0.5*vol_d)/ vol_d );

```

Rendering Passes

1. GL_LESS to capture front face

```

cgGLBindProgram(fProgramPassz);
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
//drawbrick from front
glDrawBuffer (attachmentpoints[0]);
glPushAttrib(GL_VIEWPORT_BIT|
GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glViewport(0,0,mainwindow->pViewer->w(), mainwindow->pViewer->h());
glEnable(GL_DEPTH_TEST);
glClearDepth(1.0f);
glClear(GL_DEPTH_BUFFER_BIT|GL_COLOR_BUFFER_BIT);
glDepthFunc(GL_LESS);
cgGLEnableProfile(fProfile);
Draw();
cgGLDisableProfile(fProfile);
glPopAttrib();
glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);

```

2. GL_GREATER to capture back face

```

glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, fbo);
//draw brick from back

```

```

    glDrawBuffer (attachmentpoints[1]);
    glPushAttrib(GL_VIEWPORT_BIT|
GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glViewport(0,0,mainwindow->pViewer->w(), mainwindow->pViewer->h());
    glEnable(GL_DEPTH_TEST);
    glClearDepth(0.0f);
    glClear(GL_DEPTH_BUFFER_BIT|GL_COLOR_BUFFER_BIT);
    glDepthFunc(GL_GREATER);
    cgGLEnableProfile(fProfile);
    Draw();
    cgGLDisableProfile(fProfile);
    glPopAttrib();
    glBindFramebufferEXT(GL_FRAMEBUFFER_EXT, 0);

```

3. Ray-Casting set up, need restore the viewport and enable depth test

```

//draw volumee
    glPushAttrib(GL_VIEWPORT_BIT|
GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glViewport(0,0,mainwindow->pViewer->w(), mainwindow->pViewer->h());
    glClearDepth(1.0f);
    glClear(GL_DEPTH_BUFFER_BIT);
    glDepthFunc(GL_LESS);
    glEnable(GL_DEPTH_TEST);
    glDisable(GL_BLEND);

```

```

.....
//set parameters
    cgGLBindProgram(fProgramDVR);
    cgGLEnableProfile(fProfile);
    Draw();

```

```

.....
//Disable parameters
    glPopAttrib();

```

4. Ray-Casting setup

```

float4 dst =float4(0,0,0,0);
//Determine vomlume entry position
//compute ray direction
float3 backposition = texRECT(VolExtentBack,WinPos.xy).xyz;
float3 frontposition = texRECT(VolExtentFront,WinPos.xy).xyz;

//postive or negative
float3 direction;
direction = backposition - frontposition ;
float3 position = frontposition.xyz;//+0.1*direction;

```

```
direction = normalize(direction);  
direction /= length(scale);  
direction *=stepsize;
```

```
//now step the ray, but make sure you do not step outside the volume (use the backposition  
information for this
```

```
....
```