Lecture 20

Combining 2D and 3D

- Transparency in 3D
- Particle-based rendering of cloud
- Particle-based rendering of fire
- Shadow-Map method

Transparency in 3D

- Look at transparency_1 example.
- It draws a solid cube inside a semi-transparent cube using the Phong-Shading renderer.
- In Draw() function:

```
DrawCube(5.0,0,0,1,1);
DrawCube(8.0,1,0,0,0.5);
```

- DrawCube takes four parameters, dimension, red, green, blue, and alpha.
- The program draws solid cube (alpha=1.0) first, and then semi-transparent cube (alpha=0.5).
- What if I draw the semi-transparent cube first?

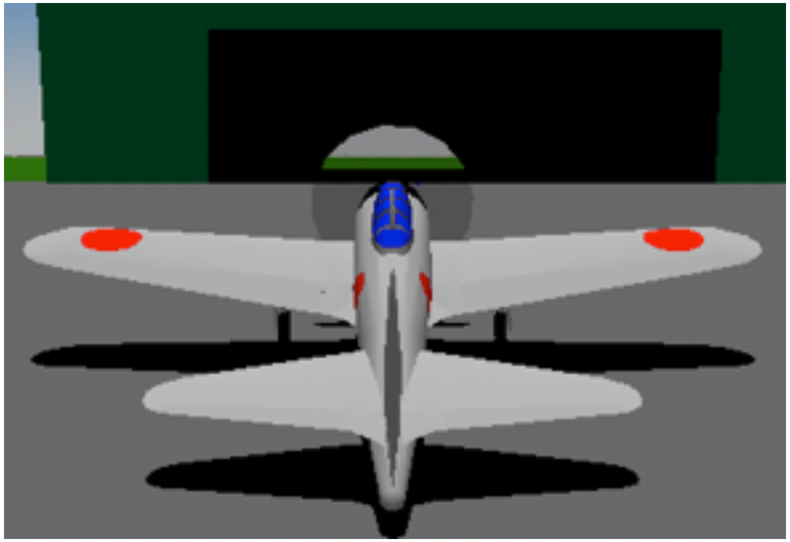
Transparency in 3D

- If a transparent object is in front of a solid object, the solid object may not be drawn.
- Even worse, it may create a peep-hole effect.

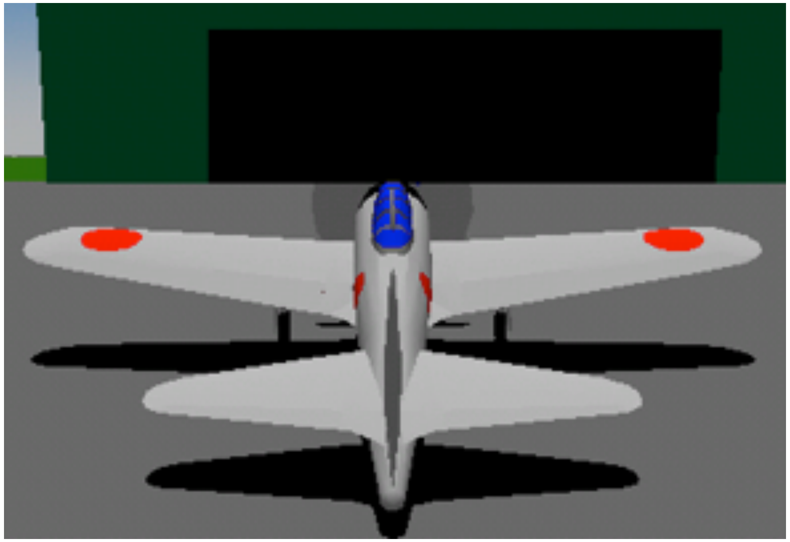
Transparency in 3D

- It requires a ray-tracing to draw a perfect image without artifacts.

- Possible Practical Solutions:
  - Sort elements based on the depth before drawing.
  - Drawing solid elements first. (Most practical)



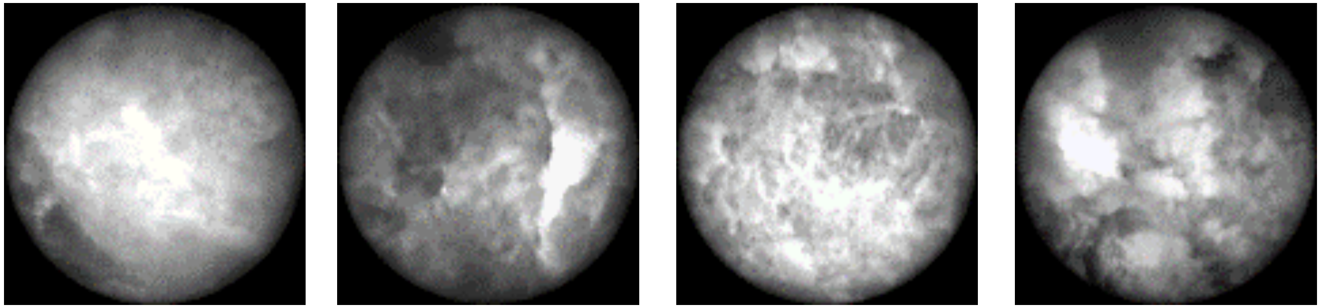
Horizon beyond the hangar is visible due to the peep-hole effect



Solid elements are drawn first.

Drawing a cloud with particle method

- Idea: Use point sprites to draw cloud, steam, and burning fire.
- Sprite patterns:



- Make a class that:
  - Generates a group of particles,
  - retains a group of particles, and
  - makes vertex attributes.
- Add particle.h

```
class ParticleGroup
{
public:
    class Particle
    {
    public:
        YsVec3 pos;
        float texCoordRange[4];
    };

    std::vector <Particle> particle;
    std::vector <float> vtx,pointSize,col,texCoordRange;

    void CleanUp(void);
    void MakeCloud(int nParticle,const YsVec3 &min,const YsVec3 &max);
    void MakeVertexBuffer(void);
};
```

Information that a particle needs to carry. To use a texture-atlas, it needs a position and texture-coordinate range.

An array of particles

Vertex Attribute Arrays

Drawing a cloud with particle method

```
#include "particle.h"
#include <stdlib.h>
void ParticleGroup::CleanUp(void)
{
    particle.clear();
    vtx.clear();
    col.clear();
    texCoordRange.clear();
}
void ParticleGroup::MakeCloud(int nParticle,const YsVec3 &min,const YsVec3 &max)
{
    particle.resize(nParticle);
    for(auto &p : particle)
    {
        double s=(double)rand()/(double)RAND_MAX;
        double t=(double)rand()/(double)RAND_MAX;
        double u=(double)rand()/(double)RAND_MAX;
        double x=min.x()*(1.0-s)+max.x()*s;
        double y=min.y()*(1.0-t)+max.y()*t;
        double z=min.z()*(1.0-u)+max.z()*u;

        p.pos.Set(x,y,z);

        // Assume 4x4 texture atlas
        s=0.25*(double)(rand()%4);
        t=0.75;
        p.texCoordRange[0]=(float)s;
        p.texCoordRange[1]=(float)t;
        p.texCoordRange[2]=(float)s+0.25;
        p.texCoordRange[3]=(float)t+0.25;
    }
}
```

The box where the particles are scattered.

Random location within the box.

Assign one of the four patterns randomly.

Drawing a cloud with particle method

Make vertex attribute arrays for rendering

```
void ParticleGroup::MakeVertexBuffer(void)
{
    vtx.resize(particle.size()*3);
    col.resize(particle.size()*4);
    pointSize.resize(particle.size());
    texCoordRange.resize(particle.size()*4);

    int idx=0;
    for(auto &p : particle)
    {
        vtx[idx*3  ]=p.pos.xf();
        vtx[idx*3+1]=p.pos.yf();
        vtx[idx*3+2]=p.pos.zf();

        col[idx*4  ]=1;
        col[idx*4+1]=1;
        col[idx*4+2]=1;
        col[idx*4+3]=0.3;

        pointSize[idx]=1.0f;

        texCoordRange[idx*4  ]=p.texCoordRange[0];
        texCoordRange[idx*4+1]=p.texCoordRange[1];
        texCoordRange[idx*4+2]=p.texCoordRange[2];
        texCoordRange[idx*4+3]=p.texCoordRange[3];

        ++idx;
    }
}
```

Drawing a cloud with particle method

- Starting from point_sprite_texture
- In Initialize(), load sprite4x4.png and create a texture from it. (The file must be in the data sub-directory.)
- Also add in Initialize(),

```
nParticle=2000;
particleGroup.MakeCloud(nParticle,YsVec3(-8,-4,-8),YsVec3(8,4,8));
particleGroup.MakeVertexBuffer();
```

- In Draw(),

```
glDrawArrays(GL_POINTS,0,particleGroup.particle.size());
```

Drawing a cloud with particle method

- Renders somewhat gaseous thing.
- But, we don’t want to see squares.
- Use alpha-cut-off in the fragment shader?

```
if(gl_FragColor.a<0.01)
{
    discard;
}
```

- Some improvement, but still not very gas-like.

Drawing a cloud with particle method

- If the pattern has only two colors, this problem can be solved by:
  - Disabling writing to Z-buffer (glDepthMask(0)), and
  - Using additive transparency (more overlapping particles make the object brighter), or
  - Using multiplicative transparency (more overlapping particles make the object darker).
- But, it makes the object somewhat monotonic.

Drawing a cloud with particle method

- Solution: Sorting the particles based on the distance from the view point.

```
void ParticleGroup::MakeVertexBuffer(const YsVec3 &viewDir,float particleSize)
{
    vtx.resize(particle.size()*3);
    col.resize(particle.size()*4);
    pointSize.resize(particle.size());
    texCoordRange.resize(particle.size()*4);
    auto idxBuf=SortIndex(viewDir);
    int idx=0;
    for(auto &sortedIdx : idxBuf)
    {
        auto &p=particle[sortedIdx];
        vtx[idx*3  ]=p.pos.xf();
        vtx[idx*3+1]=p.pos.yf();
        vtx[idx*3+2]=p.pos.zf();
        col[idx*4  ]=1;
        col[idx*4+1]=1;
        col[idx*4+2]=1;
        col[idx*4+3]=0.3;
        pointSize[idx]=particleSize;
        texCoordRange[idx*4  ]=p.texCoordRange[0];
        texCoordRange[idx*4+1]=p.texCoordRange[1];
        texCoordRange[idx*4+2]=p.texCoordRange[2];
        texCoordRange[idx*4+3]=p.texCoordRange[3];
        ++idx;
    }
}
```

Sort particles

Drawing a cloud with particle method

```
std::vector <int> ParticleGroup::SortIndex(const YsVec3 &viewDir)
{
    std::vector <int> idxBuf;
    idxBuf.resize(particle.size());

    std::vector <double> viewDist;
    viewDist.resize(particle.size());

    int idx=0;
    for(auto &p : particle)
    {
        idxBuf[idx]=idx;
        viewDist[idx]=-viewDir*p.pos;
        ++idx;
    }

    YsSimpleMergeSort <double,int> (viewDist.size(),viewDist.data(),idxBuf.data());

    return idxBuf;
}
```

Do you remember how merge-sort works?
In this function, the array elements in idxBuf are
sorted based on the elements in viewDist.

Drawing a cloud with particle method

- Modifications:
  - Increase/decrease number of particles.
  - Increase/decrease alpha.

Drawing burning fire

- This time, the particles need to move.
- Also must change color.
- First, make a program that generates and moves particles, and then change color.

Drawing burning fire

- A particle needs additional properties:
  - Velocity
  - Age (seconds after generated)
  - Life
- Two new member functions of ParticleGroup class:
  - AddFireParticle
  - Move

Drawing burning fire

```
void ParticleGroup::AddFireParticle(int nParticle)
{
    for(int i=0; i<nParticle; ++i)
    {
        double x=(double)rand()/(double)RAND_MAX;
        double y=0.0;
        double z=(double)rand()/(double)RAND_MAX;

        double vx=3.0*((double)rand()/(double)RAND_MAX-0.5);
        double vy=1.0+3.0*((double)rand()/(double)RAND_MAX);
        double vz=3.0*((double)rand()/(double)RAND_MAX-0.5);

        Particle newParticle;
        newParticle.pos.Set(x,y,z);
        newParticle.vel.Set(vx,vy,vz);
        newParticle.t=0.0;
        newParticle.tRemain=5.0+(double)rand()/(double)RAND_MAX;

        // Assume 4x4 texture atlas
        double s=0.25*(double)(rand()%4);
        double t=0.75;
        newParticle.texCoordRange[0]=(float)s;
        newParticle.texCoordRange[1]=(float)t;
        newParticle.texCoordRange[2]=(float)s+0.25;
        newParticle.texCoordRange[3]=(float)t+0.25;

        particle.push_back(newParticle);
    }
}
```

Pick a random location
0<=x<=1, 0<=z<=1, y=0

Pick a random velocity

Pick one of the four patterns in the texture atlas

Drawing burning fire

```
void ParticleGroup::Move(const double dt)
{
    for(auto &p : particle)
    {
        p.pos+=p.vel*dt;
        p.vel.AddY(0.1*dt);
        p.t+=dt;
    }

    auto newSize=particle.size();
    for(int idx=particle.size()-1; 0<=idx; --idx)
    {
        if(particle[idx].tRemain<particle[idx].t)
        {
            particle[idx]=particle[newSize-1];
            --newSize;
        }
    }
    particle.resize(newSize);
}
```

Move and give some upward acceleration

Update the age of the particle

Delete particles that come to its life.

Drawing burning fire

- In main .cpp,
  - Delete MakeCloud
  - In Interval, add AddFireParticle
  - To calculate dt, add a member variable:

    Unsigned long long lastTimer;

Drawing burning fire

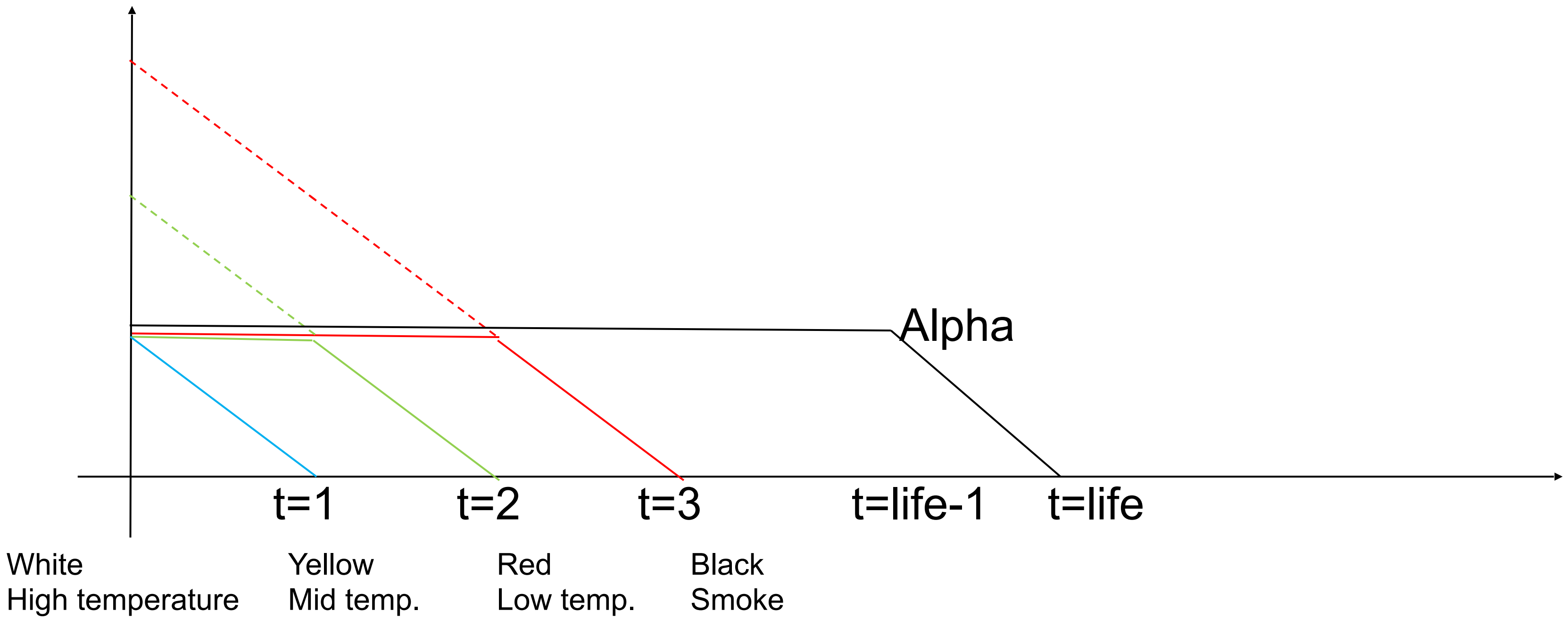
- In Interval()

```
particleGroup.AddFireParticle(10);

long long int dt=(FsSubSecondTimer()-lastTimer);
lastTimer=FsSubSecondTimer();
particleGroup.Move((double)dt/1000.0);
```

Drawing burning fire

- At this point, white smoke billows from near the origin.
- The color and alpha needs to be changed based on the time.



Draw burning fire

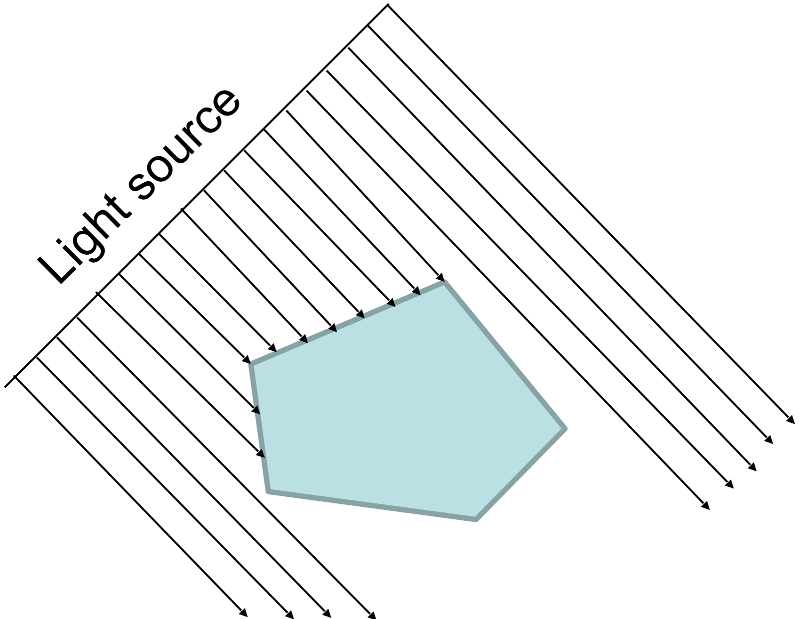
```
void ParticleGroup::MakeVertexBuffer(const YsVec3 &viewDir,float particleSize)
{
    vtx.resize(particle.size()*3);
    col.resize(particle.size()*4);
    pointSize.resize(particle.size());
    texCoordRange.resize(particle.size()*4);
    auto idxBuf=SortIndex(viewDir);
    int idx=0;
    for(auto &sortedIdx : idxBuf)
    {
        auto &p=particle[sortedIdx];
        vtx[idx*3  ]=p.pos.xf();
        vtx[idx*3+1]=p.pos.yf();
        vtx[idx*3+2]=p.pos.zf();
        float alpha=0.3*YsSmaller<float>(p.tRemain-p.t,1.0f);
        float blue=YsGreater <float> (0.0f,1.0f-p.t);
        float green=YsBound <float> (2.0-p.t,0.0f,1.0f);
        float red=YsBound <float> (3.0-p.t,0.0f,1.0f);
        col[idx*4  ]=red;
        col[idx*4+1]=green;
        col[idx*4+2]=blue;
        col[idx*4+3]=alpha;
        pointSize[idx]=particleSize;
        texCoordRange[idx*4  ]=p.texCoordRange[0];
        texCoordRange[idx*4+1]=p.texCoordRange[1];
        texCoordRange[idx*4+2]=p.texCoordRange[2];
        texCoordRange[idx*4+3]=p.texCoordRange[3];

        ++idx;
    }
}
```

Calculate color based on the age.

Shadow-map method

- Want to draw shadow more accurately, but cannot afford running ray-tracing for interactive applications.
- How?
    - Assume planar light source.
    - Create a distance map from the light source to the first intersection with an object (obstacle).
    - For each pixel, check if the distance from the light source is farther than the distance on the map, use zero diffuse & specular.



Shadow-map method

Creating a light-source to obstacle distance map

- Rendering to a texture.
- Draw the scene from the light point of view, then the depth buffer will be nothing but the distance map, that gives the distance from the light-source to the first intersection of an object.

Shadow-map method

What you need to prepare

- An additional frame buffer.
- A texture, which serves as a depth buffer for the frame buffer.

Shadow-map method

First step is to create a frame buffer. Let’s write a class called OpenGLShadowMapBuffer.

```
#ifndef SHADOW_MAP_BUFFER_IS_INCLUDED
#define SHADOW_MAP_BUFFER_IS_INCLUDED

#include "opengl_header.h"

class OpenGLShadowMapBuffer
{
public:
    GLuint frameBufferIdent;
    GLuint shadowTexIdent;

    OpenGLShadowMapBuffer();
    ~OpenGLShadowMapBuffer();
    void CleanUp(void);

    void PrepareBuffer(int wid,int hei);
};

#endif
```

The class needs to remember two identifiers, one for frame buffer, one for texture.

Shadow-map method

```
OpenGLShadowMapBuffer::OpenGLShadowMapBuffer()
{
    frameBufferIdent=0;
    shadowTexIdent=0;
    CleanUp();
}
OpenGLShadowMapBuffer::~OpenGLShadowMapBuffer()
{
    CleanUp();
}
void OpenGLShadowMapBuffer::CleanUp(void)
{
    if(0<frameBufferIdent)
    {
        glDeleteFramebuffers(1,&frameBufferIdent);
    }
    if(0<shadowTexIdent)
    {
        glDeleteTextures(1,&shadowTexIdent);
    }
}
```

Shadow-map method

```
void OpenGLShadowMapBuffer::PrepareBuffer(int shadowTexWid,int shadowTexHei)
{
   glGenTextures(1,&shadowTexIdent);
   glBindTexture(GL_TEXTURE_2D,shadowTexIdent);
#if (!defined(GL_ES) II GL_ES==0) && !defined(GL_ES_VERSION_2_0)
   glTexImage2D(GL_TEXTURE_2D,0,
      GL_DEPTH_COMPONENT32,
      shadowTexWid,shadowTexHei,0,GL_DEPTH_COMPONENT,
      GL_FLOAT,
      nullptr);
#else
   glTexImage2D(GL_TEXTURE_2D,0,
      GL_DEPTH_COMPONENT,
      shadowTexWid,shadowTexHei,0,GL_DEPTH_COMPONENT,
      GL_UNSIGNED_INT,
      nullptr);
#endif


   glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
   glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
   glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP_TO_EDGE);
   glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP_TO_EDGE);
```

ES does not have depth-component 32

ES needs to use UNSIGNED_INT

Shadow-map method

Create a frame buffer and link the texture to it.

```
    glGenFramebuffers(1,&frameBufferIdent);
    glBindFramebuffer(GL_FRAMEBUFFER,frameBufferIdent);
    glFramebufferTexture2D(GL_FRAMEBUFFER,GL_DEPTH_ATTACHMENT,GL_TEXTURE_2D,shadowTexIdent,0);
#if (!defined(GL_ES) || GL_ES==0) && !defined(GL_ES_VERSION_2_0)
    glDrawBuffer(GL_NONE);
    glReadBuffer(GL_NONE);
#else
    GLuint colorTexIdent;
    glGenTextures(1,&colorTexIdent);
    glBindTexture(GL_TEXTURE_2D,colorTexIdent);
    glTexImage2D(GL_TEXTURE_2D,0,
      GL_RGBA,shadowTexWid,shadowTexHei,0,GL_RGBA,GL_UNSIGNED_BYTE,nullptr);

    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_NEAREST);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP_TO_EDGE);
    glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP_TO_EDGE);

    glFramebufferTexture2D(GL_FRAMEBUFFER,GL_COLOR_ATTACHMENT0,GL_TEXTURE_2D,colorTexIdent,0);
#endif

    printf("%d %d\n",frameBufferIdent,shadowTexIdent);
```

Without this, frame buffer status becomes 36060, which seems to be GL_FRAMEBUFFER_INCOMPLETE_READ_BUFFER

OpenGL ES does not allow framebuffer with a depth buffer without color buffer.

Shadow-map method

```
    auto sta=glCheckFramebufferStatus(GL_FRAMEBUFFER);
    if(sta!=GL_FRAMEBUFFER_COMPLETE)
    {
        switch(sta)
        {
        case GL_FRAMEBUFFER_INCOMPLETE_ATTACHMENT:
            printf("GL_FRAMEBUFFER_INCOMPLETE_ATTACHMENT\n");
            break;
        //case GL_FRAMEBUFFER_INCOMPLETE_DIMENSIONS:
        //    printf("GL_FRAMEBUFFER_INCOMPLETE_DIMENSIONS\n");
        //    break;
#if (!defined(GL_ES) || GL_ES==0) && !defined(GL_ES_VERSION_2_0)
        case GL_FRAMEBUFFER_INCOMPLETE_READ_BUFFER:
            printf("GL_FRAMEBUFFER_INCOMPLETE_READ_BUFFER\n");
            break;
#endif
        case GL_FRAMEBUFFER_INCOMPLETE_MISSING_ATTACHMENT:
            printf("GL_FRAMEBUFFER_INCOMPLETE_MISSING_ATTACHMENT\n");
            break;
        case GL_FRAMEBUFFER_UNSUPPORTED:
            printf("GL_FRAMEBUFFER_UNSUPPORTED\n");
            break;
        default:
            printf("Unknown error %d\n",sta);
            break;
        }
        printf("Cannot generate a frame buffer.\n");
        exit(1);
    }
    glBindFramebuffer(GL_FRAMEBUFFER,0);
}
```

This macro is supposed to exist according to the OpenGL specification…

Shadow-map method

- Starting from the Phong-shading example, add shadow-map buffer.
- Add member variable:

```
OpenGLShadowMapBuffer shadowMapBuf;
```

- In Initialize() add:

```
shadowMapBuf.PrepareBuffer(1024,1024);
```

- Run and verify not getting an error message from the frame-buffer creation.

Shadow-map method

- Add a new renderer.
  - depth_verify_fragment_shader.glsl
  - depth_verify_vertex_shader.glsl
- Purpose: To verify that the depth buffer is calculated correctly.

Shadow-map method

- Vertex shader

```
#ifdef GL_ES
    #define LOWP lowp
    #define MIDP mediump
    #define HIGHP highp
#else
    #define LOWP
    #define MIDP
    #define HIGHP
#endif

attribute HIGHP vec2 vertex;
attribute HIGHP vec2 texCoord;

varying HIGHP vec2 texCoordOut;

void main()
{
    texCoordOut=texCoord;
    gl_Position=vec4(vertex,0.5,1.0);
}
```

GLSL program for OpenGL ES requires precision qualifier, which gives an error in the full-scale OpenGL. Can avoid by adding these macros.

Shadow-map method

- Fragment shader

```
#ifdef GL_ES
    #define LOWP lowp
    #define MIDP mediump
    #define HIGHP highp
#else
    #define LOWP
    #define MIDP
    #define HIGHP
#endif

uniform sampler2D texture;
varying HIGHP vec2 texCoordOut;

LOWP vec4 RainbowColor(HIGHP float t)
{
    // 0      0.25  0.5    0.75    1
    // Blue->Cyan->Green->Yellow->Red

    HIGHP float tt;
    if(t<0.0)
    {
        return vec4(0,0,1,1);
    }
    else if(t<0.25)
    {
        tt=t/0.25;
        return vec4(0,tt,1,1);
    }
    else if(t<0.5)
    {
        tt=(t-0.25)/0.25;
        return vec4(0,1,1.0-tt,1);
    }
    else if(t<0.75)
    {
        tt=(t-0.5)/0.25;
        return vec4(tt,1,0,1);
    }
    else if(t<1.0)
    {
        tt=(t-0.75)/0.25;
        return vec4(1,1.0-tt,0,1);
    }
    else
    {
        return vec4(1,0,0,1);
    }
}

void main()
{
    HIGHP float intensity;
    intensity=texture2D(texture,texCoordOut).r;
    gl_FragColor=RainbowColor(intensity);
}
```

If the texture is a depth texture, any component will return the depth.

Shadow-map method

- Add a new renderer in renderer.h and renderere.cpp
- Add two member variables:

  YsMatrix4x4 lightViewTfm,lightProjTfm
- These two matrices are needed for later rendering of the actual scene.
- Then add two new functions
  - RenderDepthBuffer

    Render to the depth buffer, and create the distance map.
  - VerifyDepthBuffer

    Directly show the distance map contents.

Shadow-map method

```
void FsLazyWindowApplication::RenderShadowBuffer(void) const
{
    glBindTexture(GL_TEXTURE_2D,0);

    int curFrameBuffer=0;
    glGetIntegerv(GL_FRAMEBUFFER_BINDING,&curFrameBuffer);

    glBindFramebuffer(GL_FRAMEBUFFER,shadowMapBuf.frameBufferIdent);

    YsAtt3 viewAtt(0,0,0);
    viewAtt.SetForwardVector(-lightDir);
```

lightDir is TO the light. To draw from the light point of view, the forward vector should be opposite of lightDir.

```
    Ys3DDrawingEnvironment drawEnv;

    int wid,hei;
    FsGetWindowSize(wid,hei);
    drawEnv.SetProjectionMode(Ys3DDrawingEnvironment::ORTHOGONAL);
    drawEnv.SetAspectRatio(1.0);
    drawEnv.SetOrthogonalProjectionHeight(10.0);
    drawEnv.SetNearFar(8.0,100.0);
```

Because 1024x1024 texture

```
    drawEnv.SetViewTarget(YsOrigin());
    drawEnv.SetViewDistance(40.0);
```

Distance should be far enough, but not too far away from the subject. The subject stays between nearz and farz in the light’s coordinate system.

```
    drawEnv.SetViewAttitude(viewAtt);

    glViewport(0,0,1024,1024);

    float viewTfm[16],projTfm[16];
    drawEnv.GetViewMatrix().GetOpenGlCompatibleMatrix(viewTfm);
    drawEnv.GetProjectionMatrix().GetOpenGlCompatibleMatrix(projTfm);
```

Shadow-map method

```
    lightViewTfm=drawEnv.GetViewMatrix();
    lightProjTfm=drawEnv.GetProjectionMatrix();

    glClear(GL_DEPTH_BUFFER_BIT);
    glEnable(GL_DEPTH_TEST);

    GLfloat projMat[16];
    drawEnv.GetProjectionMatrix().GetOpenGlCompatibleMatrix(projMat);

    GLfloat viewMat[16];
    drawEnv.GetViewMatrix().GetOpenGlCompatibleMatrix(viewMat);

    glUseProgram(plain3d.programIdent);
    glUniformMatrix4fv(plain3d.uniformProjectionPos,1,GL_FALSE,projMat);
    glUniformMatrix4fv(plain3d.uniformModelViewPos,1,GL_FALSE,viewMat);

    glEnableVertexAttribArray(plain3d.attribVertexPos);
    glEnableVertexAttribArray(plain3d.attribColorPos);

    glVertexAttribPointer(plain3d.attribVertexPos,3,GL_FLOAT,GL_FALSE,0,vtx.data());
    glVertexAttribPointer(plain3d.attribColorPos,4,GL_FLOAT,GL_FALSE,0,col.data());

    glDrawArrays(GL_TRIANGLES,0,vtx.size()/3);

    glDisableVertexAttribArray(plain3d.attribVertexPos);
    glDisableVertexAttribArray(plain3d.attribColorPos);

    glBindFramebuffer(GL_FRAMEBUFFER,curFrameBuffer);
}
```

Cache transformations for the later process.

You don’t have to clear the color buffer. Only depth matters.

The rest is same as the rendering of the actual scene.

Shadow-map method

```
void FsLazyWindowApplication::VerifyShadowMap(void) const
{
    glDisable(GL_DEPTH_TEST);

    const float rectVtx[]=
    {
        -1   ,-1,       -0.3f,-1,
        -0.3f,-0.3f,    -1   ,-0.3f
    };
    const float rectTexCoord[]=
    {
        0,0,        1,0,
        1,1,        0,1
    };

    glActiveTexture(GL_TEXTURE0);
    glBindTexture(GL_TEXTURE_2D,shadowMapBuf.shadowTexIdent);

    glUseProgram(depthVerify.programIdent);

    glEnableVertexAttribArray(depthVerify.attribVertexPos);
    glEnableVertexAttribArray(depthVerify.attribTexCoordPos);

    glUniform1i(depthVerify.uniformTexturePos,0);
    glVertexAttribPointer(depthVerify.attribVertexPos,2,GL_FLOAT,GL_FALSE,0,rectVtx);
    glVertexAttribPointer(depthVerify.attribTexCoordPos,2,GL_FLOAT,GL_FALSE,0,rectTexCoord);
    glDrawArrays(GL_TRIANGLE_FAN,0,4);

    glDisableVertexAttribArray(depthVerify.attribVertexPos);
    glDisableVertexAttribArray(depthVerify.attribTexCoordPos);
}
```

Shadow-map method

- At the beginning of Draw() function, call RenderShadowBuffer().
- Before FsSwapBuffers(), call VerifyShaowMap()

- Make sure the distance map makes sense.

Shadow-map method

- Finding the light-source to obstacle distance.
- For each pixel, need to find the screen coordinate (which is same as the texture coordinate) in the light's point of view.
- The transformation for the light's point of view:

  $q=P_L*V_L*M*p$

  where P_L, V_L, and M are the projection matrix, view matrix, and modeling matrix (optional) used for rendering the distance map, respectively.
- The transformation for rendering the actual scene:

  $p'=P*V*M*p$

  where P, V, and M are the projection matrix, view matrix, and modeling matrix (optional).

Shadow-map method

- A 3D renderer (like Phong-shading renderer) knows P (projection matrix), and V*M (model-view matrix).
- To find q in the 3D renderer,

  $q=P_L*V_L*V^{-1}*V*M*p$
- Therefore, $T_L=P_L*V_L*V^{-1}$ must be given as an additional uniform. (Shadow-map transformation)
- Also additional varying q (position in the light-coordinate) is needed.

Shadow-map method

- Changes in the Phong-shading vertex shader.

```
attribute vec3 vertex;
attribute vec3 normal;
attribute vec4 color;

uniform mat4 projection,modelView;
uniform vec3 lightDir;
uniform float ambient;
uniform float specularIntensity;
uniform float specularExponent;

uniform mat4 shadowMapTfm;

varying vec4 colorOut;
varying vec3 normalOut;
varying vec3 viewDirOut;
varying vec4 shadowMapCoord;

void main()
{
   vec4 posInView=modelView*vec4(vertex,1.0);
   viewDirOut=-normalize(posInView.xyz);
   normalOut=normalize((modelView*vec4(normal,0.0)).xyz);
   colorOut=color;
   gl_Position=projection*modelView*vec4(vertex,1.0);

   shadowMapCoord=shadowMapTfm*modelView*vec4(vertex,1.0);
}
```

Shadow-map method

- Changes in the Fragment shader

```
varying vec4 colorOut;
varying vec3 normalOut;
varying vec3 viewDirOut;

uniform vec3 lightDir;
uniform float ambient;
uniform float specularIntensity;
uniform float specularExponent;

uniform sampler2D shadowMapTexture;
varying vec4 shadowMapCoord;

vec4 RainbowColor(float t)
{
   (Copied from the depth_verify_fragment_shader.glsl)
}
```

Shadow-map method

```
void main()
{
    vec3 lit=normalize(lightDir);

    float diffuse=max(0.0,dot(normalOut,lit));

    vec3 midDir=normalize(viewDirOut+lightDir);
    float specular=specularIntensity*pow(dot(midDir,normalOut),specularExponent);


    vec3 shadowCoordTfm=shadowMapCoord.xyz/shadowMapCoord.w;
    shadowCoordTfm=(shadowCoordTfm+vec3(1,1,1))/2.0;
    if(0.0<shadowCoordTfm.x && shadowCoordTfm.x<1.0 &&
       0.0<shadowCoordTfm.y && shadowCoordTfm.y<1.0)
    {
        float depth;
        depth=texture2D(shadowMapTexture,shadowCoordTfm.xy).r;
        gl_FragColor=RainbowColor(depth);
    }
    else
    {
        gl_FragColor=vec4(1,0,1,1);
    }


    //gl_FragColor=vec4(colorOut.rgb*(ambient+diffuse),colorOut.a)
    //          +vec4(specular,specular,specular,0.0);
}
```

Assigns a color depending on the distance from the light-source

Shadow-map method

- In Draw() function,

```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D,shadowMapBuf.shadowTexIdent);
glUniform1f(phong3d.uniformShadowMapTexturePos,0);

YsMatrix4x4 viewInv=drawEnv.GetViewMatrix();
viewInv.Invert();
YsMatrix4x4 shadowTfm=lightProjTfm*lightViewTfm*viewInv;
GLfloat shadowTfmf[16];
shadowTfm.GetOpenGlCompatibleMatrix(shadowTfmf);
glUniformMatrix4fv(phong3d.uniformShadowMapTfmPos,1,GL_FALSE,shadowTfmf);
```

Shadow-map method

- Can verify light-source to obstacle distances are correctly mapped.

Shadow-mapping method

- Finally, the fragment shader is modified so that diffuse and specular are set to zero if the vertex to light-source distance is greater than light-source to first obstacle distance.

```
vec3 shadowCoordTfm=shadowMapCoord.xyz/shadowMapCoord.w;
shadowCoordTfm=(shadowCoordTfm+vec3(1,1,1))/2.0;
if(0.0<shadowCoordTfm.x && shadowCoordTfm.x<1.0 &&
   0.0<shadowCoordTfm.y && shadowCoordTfm.y<1.0)
{
    float depth;
    depth=texture2D(shadowMapTexture,shadowCoordTfm.xy).r;
    if(shadowCoordTfm.z>depth)
    {
        diffuse=0.0;
        specular=0.0;
    }
}
```

Shadow-map method

- What to do with the moire pattern?
- Coming from the numerical error.