Sogang University Computer Graphics Lab.

MULTI-PASS VS SINGLE-PASS CUBEMAP

2008.4

Contents

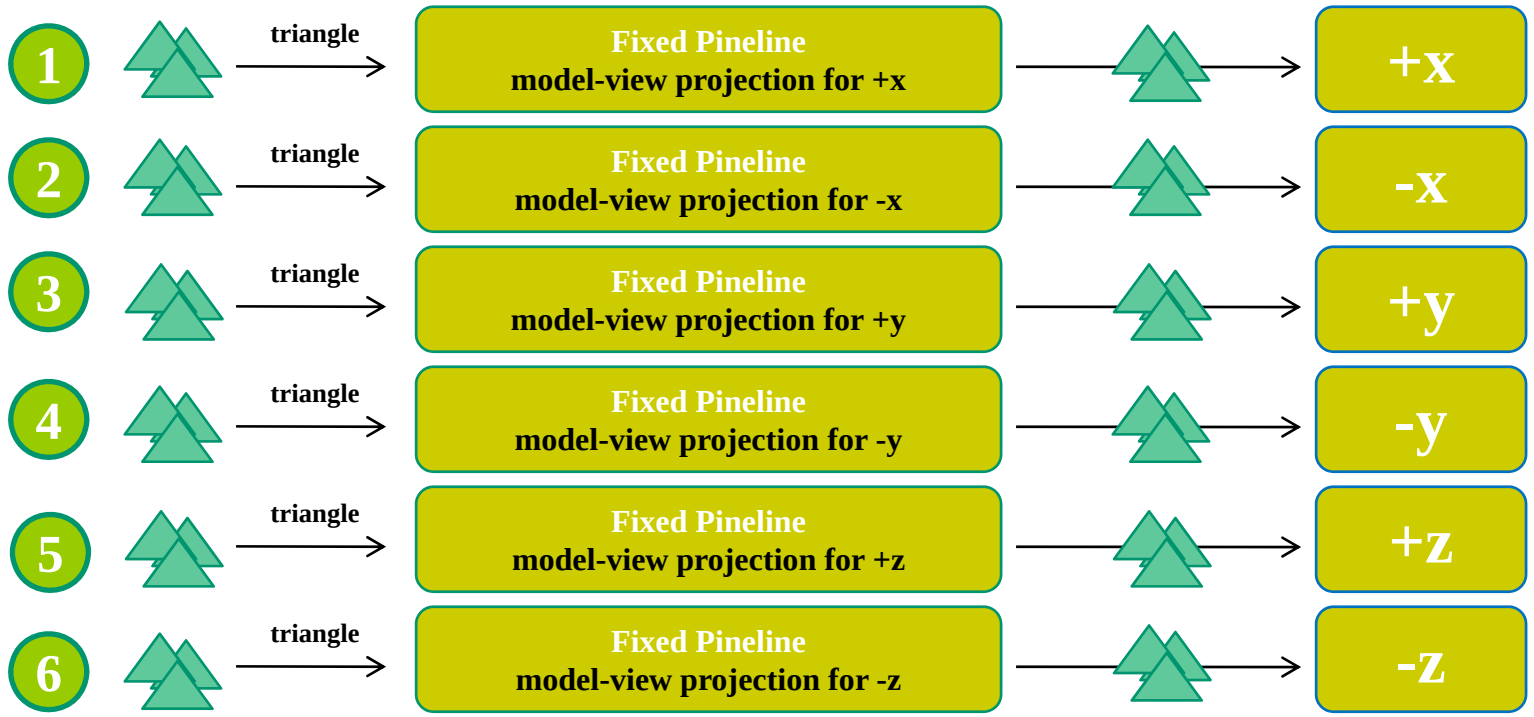
- Purpose
- Multi-Pass Cubemap
- Single-Pass Cubemap
- Reflection Mapping
- Test Result
- Conclusion

Purpose

- Implement real-time cubemap creation and reflection mapping.
- Implement multi-pass cubemap using OpenGL fixed pipeline.
- Implement single-pass cubemap using geometry program.
- Use framebuffer object and renderbuffer of Nvidia Extension.
- Use assembly language on Nvidia G8x specification.
- Compare single-pass cubemap with multi-pass cubemap performance.

Multi-Pass Cubemap (1/3)

- Rendering function is called six times for making each face of cubemap.
- Since each face of cubemap is created sequentially, only one depth buffer is required.
- Use OpenGL fixed pipeline.



Multi-Pass Cubemap (2/3)

- OpenGL Code (framebuffer object, cubemap texture initializing)

```
GLuint six_fb, six_pass_cube_tex;
GLuint depth_rb;

// framebuffer object, cubemap texture, depth buffer generate
glGenFramebuffersEXT( 1, &six_fb );
glGenTextures( 1, & six_pass_cube_tex);
glGenRenderbuffersEXT( 1, &depth_rb );

glBindTexture( GL_TEXTURE_CUBE_MAP_ARB, six_pass_cube_tex);
glTexParameteri( GL_TEXTURE_CUBE_MAP_ARB, GL_TEXTURE_MIN_FILTER, GL_NEAREST );
glTexParameteri( GL_TEXTURE_CUBE_MAP_ARB, GL_TEXTURE_MAG_FILTER, GL_NEAREST );
glTexParameteri( GL_TEXTURE_CUBE_MAP_ARB, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE );
glTexParameteri( GL_TEXTURE_CUBE_MAP_ARB, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE );
glTexParameteri( GL_TEXTURE_CUBE_MAP_ARB, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE );

// create texture for each face of cubemap.
for ( int i = 0; i < 6; ++i ) {
                glBindTexture( GL_TEXTURE_CUBE_MAP_ARB, six_pass_cube_tex);
                glTexImage2D( GL_TEXTURE_CUBE_MAP_POSITIVE_X_ARB + i, 0,
                                GL_RGB, fbWidth, fbHeight, 0, GL_RGB, GL_UNSIGNED_BYTE, NULL );
}

// render buffer creation for depth buffer
glBindRenderbufferEXT( GL_RENDERBUFFER_EXT, depth_rb );
glRenderbufferStorageEXT( GL_RENDERBUFFER_EXT, GL_DEPTH_COMPONENT32, fbWidth, fbHeight );
```

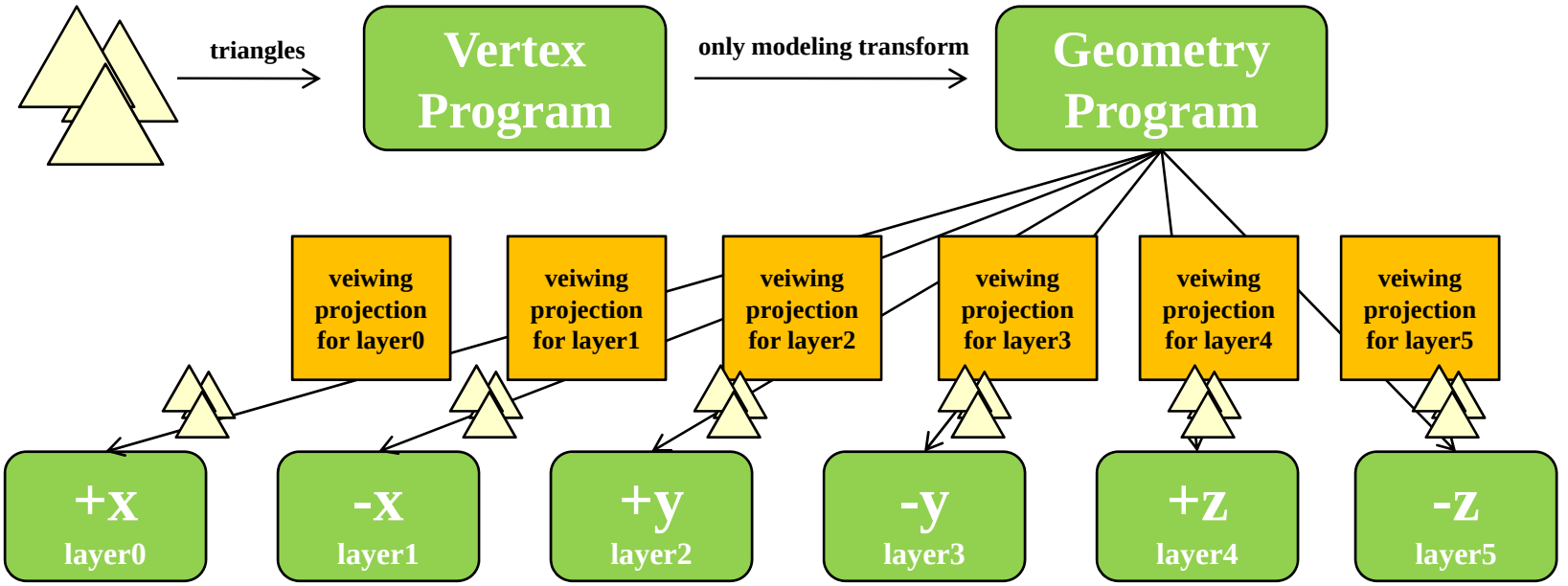
Multi-Pass Cubemap (3/3)

- OpenGL Code (cubemap creation)

```
// bind frame buffer object
glBindFramebufferEXT( GL_FRAMEBUFFER_EXT, six_fb );

// make each face of cubemap
for ( int face = 0; face < 6; ++face ) {

        glMatrixMode( GL_PROJECTION );
        glLoadIdentity();

        // set model-view projection matrix.
        glMatrixMode( GL_MODELVIEW );
        glLoadMatrixf( m_CubeMapProjectionMatrix[face].matrix );

        glFramebufferTexture2DEXT( GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT,
                GL_TEXTURE_CUBE_MAP_POSITIVE_X_ARB + face, six_pass_cube_tex, 0 );
        glFramebufferRenderbufferEXT( GL_FRAMEBUFFER_EXT, GL_DEPTH_ATTACHMENT_EXT,
                GL_RENDERBUFFER_EXT, depth_rb );

        glDrawBuffer( GL_COLOR_ATTACHMENT0_EXT );

        ……. Draw objects  ……..

        glFlush();

}
```

Single-Pass Cubemap (1/5)

- Draw six faces of cubemap at once.
- Use geometry program.
- Need six depth buffer. (EXT_texture_array is used)



Single-Pass Cubemap (2/5)

- OpenGL Code (cubemap creation)

```
glBindTexture( GL_TEXTURE_2D_ARRAY_EXT, depth_buffer_array );
glTexEnvf( GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE );
glTexParameteri( GL_TEXTURE_2D_ARRAY_EXT, GL_TEXTURE_MIN_FILTER, GL_NEAREST );
glTexParameteri( GL_TEXTURE_2D_ARRAY_EXT, GL_TEXTURE_MAG_FILTER, GL_NEAREST );
glTexImage3D( GL_TEXTURE_2D_ARRAY_EXT, 0, GL_DEPTH_COMPONENT24, 256, 256, 6, 0,
                GL_DEPTH_COMPONENT, GL_FLOAT, NULL );
```

```
glBindFramebufferEXT( GL_FRAMEBUFFER_EXT, single_fb );
glFramebufferTextureEXT( GL_FRAMEBUFFER_EXT, GL_COLOR_ATTACHMENT0_EXT, one_pass_cube_tex_array, 0 );
// depth buffer setting.
glFramebufferTextureEXT( GL_FRAMEBUFFER_EXT, GL_DEPTH_ATTACHMENT_EXT, depth_buffer_array, 0 );

glDrawBuffer( GL_COLOR_ATTACHMENT0_EXT );
glEnable( GL_GEOMETRY_PROGRAM_NV );
glEnable( GL_VERTEX_PROGRAM_NV );
glBindProgramNV( GL_GEOMETRY_PROGRAM_NV, m_OnepassGeometryProgramID );
glBindProgramNV( GL_VERTEX_PROGRAM_NV, m_OnepassVertexProgramID );

int index = 0;
// set projection matrix for each face of cubemap.
for ( int i = 0; i < 6; ++i ) {
  glProgramLocalParameter4fARB( GL_GEOMETRY_PROGRAM_NV, index++,  m_CubeMapProjectionMatrix[i][0],
                m_CubeMapProjectionMatrix[i][4],  m_CubeMapProjectionMatrix[i][8],  m_CubeMapProjectionMatrix[i][12] );
  glProgramLocalParameter4fARB( GL_GEOMETRY_PROGRAM_NV, index++,  m_CubeMapProjectionMatrix[i][1],
                m_CubeMapProjectionMatrix[i][5],  m_CubeMapProjectionMatrix[i][9],  m_CubeMapProjectionMatrix[i][13] );
  glProgramLocalParameter4fARB( GL_GEOMETRY_PROGRAM_NV, index++,  m_CubeMapProjectionMatrix[i][2],
                m_CubeMapProjectionMatrix[i][6],  m_CubeMapProjectionMatrix[i][10],  m_CubeMapProjectionMatrix[i][14] );
  glProgramLocalParameter4fARB( GL_GEOMETRY_PROGRAM_NV, index++,  m_CubeMapProjectionMatrix[i][3],
                m_CubeMapProjectionMatrix[i][7],  m_CubeMapProjectionMatrix[i][11],  m_CubeMapProjectionMatrix[i][15] );
}
… Draw Object …
```

Single-Pass Cubemap (3/5)

- Vertex Program

  - Vertex is passed to geometry program after modeling transform is applied.

```
!!NVvp4.0

#-------------------------------------------------------------------#
# Input Binding
#-------------------------------------------------------------------#

# must be setted only model transform matrix without view transform matrix
PARAM modelMatrix[] = { state.matrix.modelview.row[0..3] };

#-------------------------------------------------------------------#
# Code
#-------------------------------------------------------------------#

DP4.F result.position.x, modelMatrix[0], vertex.position;
DP4.F result.position.y, modelMatrix[1], vertex.position;
DP4.F result.position.z, modelMatrix[2], vertex.position;
DP4.F result.position.w, modelMatrix[3], vertex.position;

MOV.F result.texcoord, vertex.texcoord;
MOV.F result.color, vertex.color;

END
```

Single-Pass Cubemap (4/5)

- Geometry Program (1/2)
  - Triangle is projected to each face of cubemap.
    (1 Triangle Input → 6 Triangle Output)
  - Use 0~5 layer for each face of cubemap.

```
!!NVgp4.0

#-----------------------------------------------------------------------------#
# Option
#-----------------------------------------------------------------------------#
PRIMITIVE_IN TRIANGLES;
PRIMITIVE_OUT TRIANGLE_STRIP;
VERTICES_OUT 18;

#-----------------------------------------------------------------------------#
# Program Parameter.
#-----------------------------------------------------------------------------#
# view projection matrix for each cube face
PARAM vpMatrix[] = { program.local[0..23] };

#-----------------------------------------------------------------------------#
# Temp
#-----------------------------------------------------------------------------#

INT TEMP nCount;
INT TEMP index;
INT TEMP nLayer;
```

Single-Pass Cubemap (5/5)

- Geometry Program (2/2)

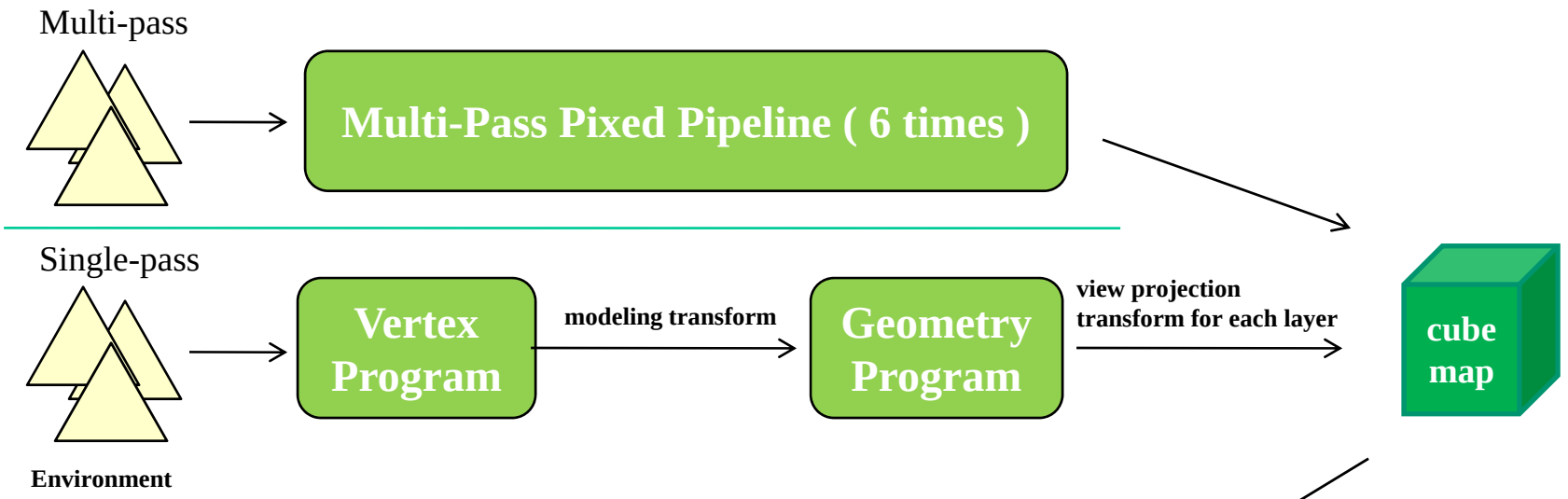
```
#-------------------------------------------------------------------------#
# Code
#-------------------------------------------------------------------------#

MOV.S nLayer, 0;
MOV.S index.x, 0;

# six times loop for each cube face
REP.S {6};

  # projection matrix index for each layer
  MUL.S index.x, nLayer.x, {4};

  #  vertex1 projection.
  DP4.F result.position.x, vpMatrix[index.x + 0], vertex[0].position;
  DP4.F result.position.y, vpMatrix[index.x + 1], vertex[0].position;
  DP4.F result.position.z, vpMatrix[index.x + 2], vertex[0].position;
  DP4.F result.position.w, vpMatrix[index.x + 3], vertex[0].position;
  MOV.F result.texcoord[0], vertex[0].texcoord[0];
  MOV.F result.color, vertex[0].color;
  MOV.S result.layer.x, nLayer.x;
  EMIT;
```

```
  #  vertex1 projection.
  DP4.F result.position.x, vpMatrix[index.x + 0], vertex[1].position;
  DP4.F result.position.y, vpMatrix[index.x + 1], vertex[1].position;
  DP4.F result.position.z, vpMatrix[index.x + 2], vertex[1].position;
  DP4.F result.position.w, vpMatrix[index.x + 3], vertex[1].position;
  MOV.F result.texcoord[0], vertex[1].texcoord[0];
  MOV.F result.color, vertex[1].color;
  MOV.S result.layer.x, nLayer.x;
  EMIT;

  DP4.F result.position.x, vpMatrix[index.x + 0], vertex[2].position;
  DP4.F result.position.y, vpMatrix[index.x + 1], vertex[2].position;
  DP4.F result.position.z, vpMatrix[index.x + 2], vertex[2].position;
  DP4.F result.position.w, vpMatrix[index.x + 3], vertex[2].position;
  MOV.F result.texcoord[0], vertex[2].texcoord[0];
  MOV.F result.color, vertex[2].color;
  MOV.S result.layer.x, nLayer.x;
  EMIT;
  ENDPRIM;

  ADD.S nLayer.x, nLayer.x, {1};

ENDREP;
END
```

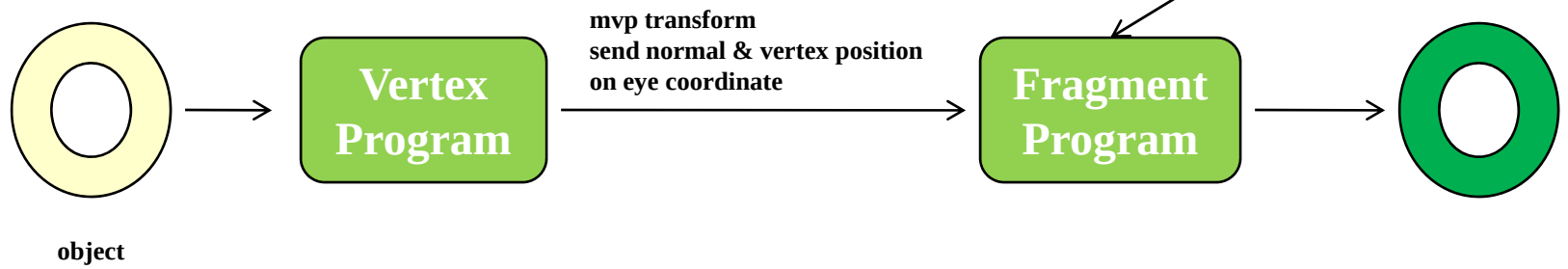
Reflection Mapping (1/3)

- Reflection mapping flow

1. Cubemap creation



2. Reflection mapping



Reflection Mapping (2/3)

- Vertex Program
  - Vertex and normal are passed to fragment program after model-view transform is applied.

```
!!NVvp4.0

#-------------------------------------------------------------------#
# Program Parameter
#-------------------------------------------------------------------#

PARAM mvpMatrix[] = { state.matrix.mvp };
PARAM mvMatrix[] = { state.matrix.modelview.row[0..3] };

PARAM mvInvTransMatrix[] = { state.matrix.modelview.invtrans };

#-------------------------------------------------------------------#
# Output Bindings
#-------------------------------------------------------------------#

# for vertex normal info ( eye coordinate )
OUTPUT vertexNormalTexure = result.texcoord[6];

# for vertex position info ( eye coodinate )
OUTPUT vertexPositionTexure = result.texcoord[7];

#-------------------------------------------------------------------#
# Code
#-------------------------------------------------------------------#
```

```
# vertex normal on eye coordinate
DP3 vertexNormalTexure.x, mvInvTransMatrix[0], vertex.normal;
DP3 vertexNormalTexure.y, mvInvTransMatrix[1], vertex.normal;
DP3 vertexNormalTexure.z, mvInvTransMatrix[2], vertex.normal;

# vertex position on eye coordinate
DP4 vertexPositionTexure.x, mvMatrix[0], vertex.position;
DP4 vertexPositionTexure.y, mvMatrix[1], vertex.position;
DP4 vertexPositionTexure.z, mvMatrix[2], vertex.position;
DP4 vertexPositionTexure.w, mvMatrix[3], vertex.position;

# to CC
DP4 result.position.x, mvpMatrix[0], vertex.position;
DP4 result.position.y, mvpMatrix[1], vertex.position;
DP4 result.position.z, mvpMatrix[2], vertex.position;
DP4 result.position.w, mvpMatrix[3], vertex.position;

MOV result.color, vertex.color;

END
```

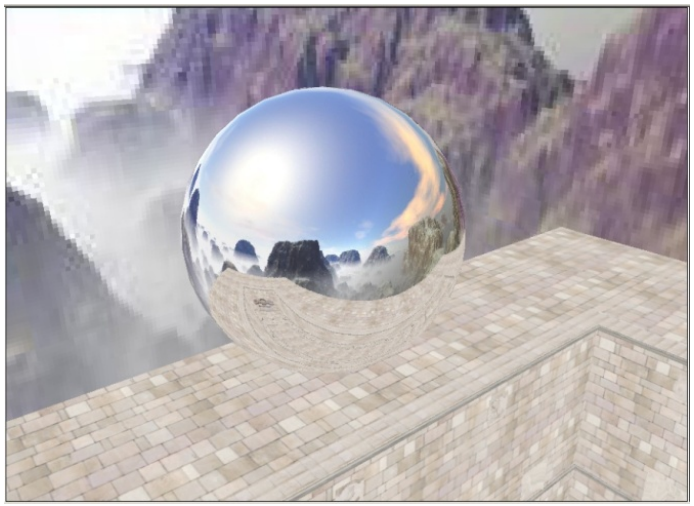
Reflection Mapping (3/3)

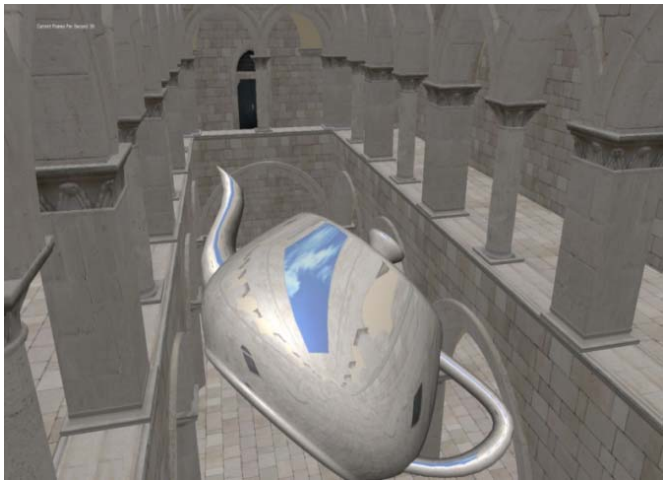
- Fragment Program
  - calculate reflection vector on eye coordinates and use cubemap.

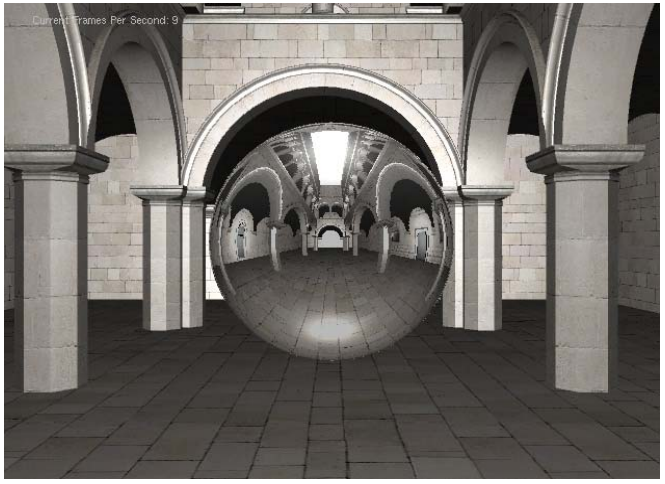
```
!!NVfp4.0

#-------------------------------------------------------------------------------#
# Input Binding
#-------------------------------------------------------------------------------#

ATTRIB inColor = fragment.color.primary;                    #  Object color
ATTRIB vertexNormal = fragment.texcoord[6];                 #  Vertex Normal ( eye Coordinates )
ATTRIB vertexPosition = fragment.texcoord[7];               #  Vertex Position ( eye Coordinates )

#-------------------------------------------------------------------------------#
# Temporary Register
#-------------------------------------------------------------------------------#

TEMP nVertexNormal, nVertexPosition, texelColor;            #  CubeMap Color
TEMP reflectionVec;                                         #  reflection vector of view vector by vertex normal

#-------------------------------------------------------------------------------#
# Code
#-------------------------------------------------------------------------------#

NRM nVertexNormal, vertexNormal;                            #  normal
NRM nVertexPosition, vertexPosition;                        #  view vector
RFL reflectionVec, nVertexNormal, -nVertexPosition;         #  reflection vector of view vector by vertex normal
TEX texelColor, reflectionVec, texture, CUBE;               #  get tex color from cube map

MOV result.color, texelColor;
END
```

Test Result (1/4)

- Result snapshot







Test Result (2/4)

- Test Environment
  - When each frame is rendered, cubemap is also re-created.
  - Since CPU and GPU are running in parallel, CPU time check routine is not correct.
  - For time checking, Use GPU Time check routine. (glBeginQuery, glEndQuery)
  - Nvidia Geforce 8800 GTX.

```
GLuint query[2];
GLint available = 0;
GLuint64EXT timeElapsed = 0;

glGenQueries( 1, query );
glBeginQuery( GL_TIME_ELAPSED_EXT, query[0] );

……. Draw Scene ……..

glEndQuery( GL_TIME_ELAPSED_EXT );

while (!available) {
        glGetQueryObjectiv( query[0], GL_QUERY_RESULT_AVAILABLE, &available );
}
 glGetQueryObjectui64vEXT( query[0], GL_QUERY_RESULT, &timeElapsed );
```

Test Result (3/4)

- When glVertex function series is used.
  - CPU→GPU data transmission is slow.
  - Data transmission is bottleneck of whole processing.
  - This case, single-pass cubemap is more fast.
    (single-pass : 1 data transmission + geometry program overhead
    < multi-pass : 6 data transmission)
- When glDrawElements function series is used.
  - CPU→GPU data transmission is fast.
  - Geometry program is more overhead than data transmission.
  - This case, multi-pass cubemap is more fast.

	glVertex	glDrawElements
Single Pass	30fps	30fps
Multi Pass	10fps	50fps

Sponza Data (60K Vertex) on WindowXP, Nvidia G880 GTX

Test Result (4/4)

- Unexpectedly, performance of multi-pass was degraded in window vista.
  - we guess that there is some bottleneck using glDrawElements in vista.

Window XP

	glVertex	glDrawElements
Single Pass	30fps	30fps
Multi Pass	10fps	50fps

Window Vista

	glVertex	glDrawElements
Single Pass	30fps	30fps
Multi Pass	10fps	29fps

Sponza Data (60K Vertex)

Conclusion

- It is known that performance of geometry program is degraded as more vertices are output.

  [reference]

  - http://www.gamedev.net/community/forums/topic.asp?topic_id=491090
  - http://www.gpgpu.org/forums/viewtopic.php?t=4851&sid=f395110947edbe71c4512b1f68eea063
  - http://www.devmaster.net/forums/showthread.php?t=12007
  - http://developer.download.nvidia.com/whitepapers/2007/SDK10/Cloth.pdf (5 page)

- Our result is also equal to above facts.
- In current geometry program, single-pass cubemap is slower than multi-pass cubemap in general (glDrawElements function series is used in general)