KỸ THUẬT BUMP MAPPING

Bump Mapping Techniques

Nguyễn Văn Trường

Trung tâm Công nghệ Mô phỏng, Học viện KTQS

Tóm tắt:

Gần như bất cứ ai có kiến thức về đồ họa 3D, hoặc các nhà phát triển phần mềm đồ họa hoặc họa sỹ máy tính, sẽ có những hiểu biết cơ bản về kỹ thuật bump mapping. Dùng việc đảo vector chỉ phương để tính toán ánh sáng, làm tăng độ chi tiết, mà không cần thêm chi tiết hình học vào lưới miêu tả đối tượng. Bump mapping hiện nay đang trở thành khuôn mẫu hiển thị 3D thời gian thực, tuy nhiên, nhiều kỹ thuật khác nhau có thể sử dụng để thực hiện bump mapping. Các kỹ thuật này có thể là đơn giản hoặc phức tạp, và thực sự khó khăn cho người lập trình đồ họa thiếu kinh nghiệm. Mục đích của bài báo này sẽ giới thiệu những kỹ thuật Bump mapping trong xử lý đồ họa 3D thời gian thực.

Từ khóa: Bump mapping, Normal map, per-pixel lighting

Almost anyone who has some knowledge of 3D graphics, either as a developer or as an artist, will have at least a basic idea of what bump mapping is. Bump mapping is a lighting technique that perturbs the normal vector of a surface on a per-pixel basis, using a texture map as input to model the perturbations. By using the perturbed normal for the lighting calculations, the apparent detail of the surface is greatly enhanced, without having to add extra geometric detail to the mesh.Bump mapping has been ubiquitous in offline rendering systems for ages, and is now also becoming the norm for realtime rendering engines. In the context of realtime rendering, however, many different techniques can be used to implement bump mapping. These techniques range from very simple to very advanced, and not all of them may be familiar to inexperienced graphics programmers. The goal of this article is to introduce novice bump mappers to the most important realtime techniques, to explain the buzzwords that surround realtime bump mapping, and maybe also to dispell a myth or two.

1. Đặt vấn đề

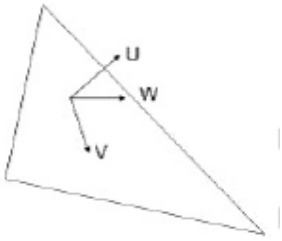
Hiện nay hiệu ứng chiếu sáng trên từng điểm ảnh được sử dụng khá phổ biến trong các sản phẩm mô phỏng nhằm tăng cường chất lượng đồ họa mô phỏng. Thay vì chiếu sáng theo từng đỉnh vertex, per-pixel lighting cho chất lượng đồ họa cao hơn hẳn do có thể áp dụng nhiều thuật toán mới trong đồ họa 3 chiều như bump bề mặt bằng normal map, phản chiếu bề mặt bằng specular map…

Ứng dụng khi tự thực hiện chiếu sáng trên điểm ảnh bằng Shaders phải giải quyết các tất cả các vấn đề về chiếu sáng như diffuse lighting, specular lighting… Diffuse lighting trong các thư viện chủ yếu sử dụng bump bằng normal map và specular lighting chủ yếu sử dụng specular map, 2 thuật toán chiếu sáng này sẽ được trình bày kỹ ở phần này.

2. Cơ sở lý thuyết

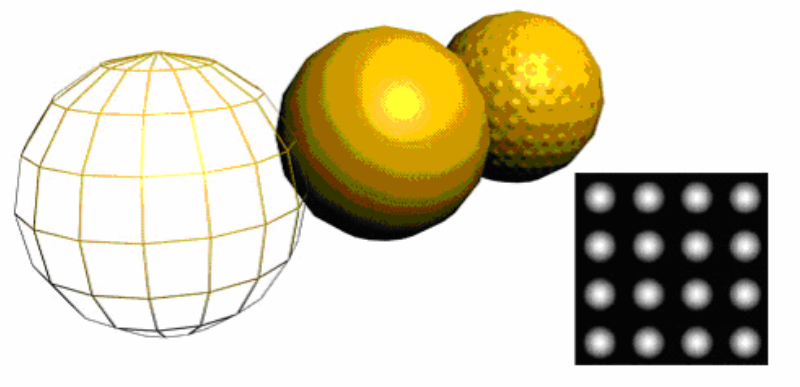
Ở phần này sẽ đề cập chi tiết vào qui trình chiếu sáng trên điểm ảnh được các thư viện đồ họa hiện nay hỗ trợ. Qui trình chiếu sáng trên điểm ảnh chủ yếu phân ra làm 2 công đoạn riêng biệt: thực hiện tính toán màu chính (diffuse color) và tính toán màu phản chiếu (specular color).

- Tính toán màu diffuse (có bump bề mặt bằng normal map) Trước khi đi chi tiết vào thuật toán ta cần xem qua 1 số khái niệm mới dùng trong phần này Không gian tiếp tuyến của vật thể (tangent space). Tọa độ texture tại mỗi đỉnh (vertex) hình thành một hệ trục tọa độ 3 chiều với trục U (tiếp tuyến), trục W (pháp tuyến) và trục V (binormal = U x W). Hệ trục tọa độ này gọi là không gian tiếp tuyến hay không gian texture của vật thể tại các đỉnh (vertex).

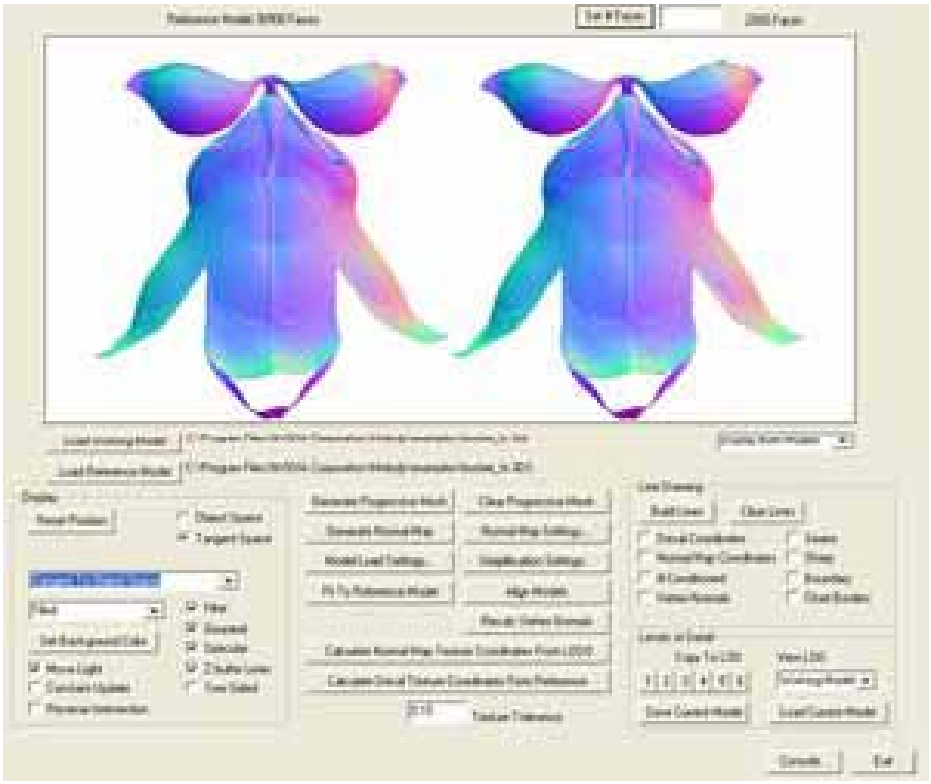


Không gian tiếp tuyến

- Normal map, Normal map là một texture nhưng có đặc tính khá đặc biệt, thay vì chứa thông tin về điểm màu như texture thông thường, normal map lai chứa thông tin về không gian tiếp tuyến (tangent space) hay không gian texture (texture space) của vật thể, hay nói cách khác nếu các điểm ảnh của texture biểu diễn màu sắc của vật thể tại 1 điểm thì normal map sẽ biểu diễn không gian tiếp tuyến của vật thể tại điểm đó. Mỗi điểm ảnh của normal map có định dạng là RGBA trong đó 3 thành phần RGB có giá trị [0..1] được ánh xạ từ 3 trục U, V, W có giá trị trong khoảng [-1, 1]. Normal map có thể tạo ra bằng 2 cách, dùng height map (texture dạng graycale chứa thông tin độ sâu về bề mặt của vật thể trong đó màu sáng hơn biểu thị độ cao lớn hơn). Cách thứ 2 phức tạp hơn do phải tạo thêm 1 vật thể khác có độ chi tiết cao hơn, sau đó ta so sánh sự khác nhau giữa 2 vật thể để tạo ra normal map (quá trình này có thể được thực hiện bằng tool Melody của NVidia).



Hình 26. Tạo normal map từ height map

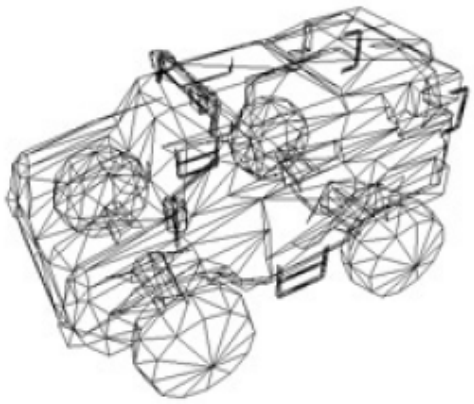


Hình 26. Tạo normal map từ vật thể có độ chi tiết cao hơn bằng Melody (NVidia)

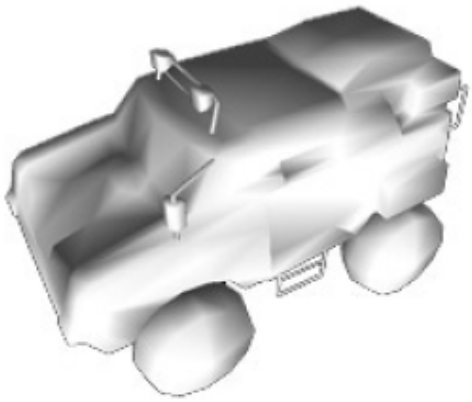
- Bump bề mặt sử dụng normal map, Bump bề mặt chủ yếu được thực hiện trên Pixel Shader cho từng điểm ảnh. Thuật toán này sử dụng giá trị normal trong normal map để xác định mức độ của ánh sáng tác động vào điểm ảnh đó bằng cách nhân tích vô hướng giá trị normal trên với vector hướng ánh sáng trong không gian tiếp tuyến. Sau đó giá trị này được nhân với màu sắc của vertex và màu lấy mẫu từ texture để tính ra màu diffuse (màu của vertex được tính trong Vertex Shader)

  Trong đó:

- Normal. Vector pháp tuyến (normal vector) tại điểm đó, normal vector có được do lấy mẫu từ normal map.
- Light vector. Vector hướng ánh sáng trong không gian tiếp tuyến (tangent space), vector này được tính trong Vertex Shader và được truyền vào Pixel Shader để sử dụng. vertex color. Màu của vertex sau khi thực hiện chiếu sáng trên vertex (pervertex lighting) trong Vertex Shader. texture color. Màu của texture chính, có được do lấy mẫu texture.



Hình 26. Dữ liệu vertex trong bộ nhớ (được vẽ dưới dạng wireframe)

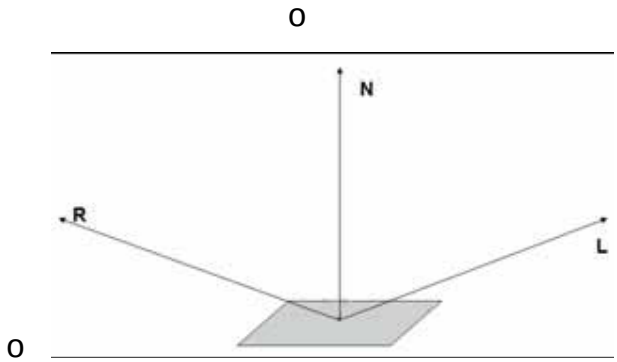


Hình 27. Chiếu sáng trên từng đỉnh bằng Vertex Shader

Hình 27. Chiếu sáng trên từng pixel (sử dụng tích vô hướng giữa normal và vector hướng ánh sáng).

Hình 28. Sau khi kết hợp với lấy mẫu từ texture chính.

- Tính toán màu specular (sử dụng specular map), specular map là texture dạng grayscale, specular map có tác dụng cho biết vùng nào của vật thể phản chiếu nhiều ánh sáng vùng nào phản chiếu ít ánh sáng (tương ứng với màu trong specular map từ sáng tới tối).
- Tính độ phản chiếu của ánh sánh, Độ phản chiếu (phản xạ) của ánh sáng trên vật thể thì phụ thuộc vị trí của mắt (hay camera). Khi mắt nằm ngay trên đường phản xạ của ánh sáng thì mắt sẽ nhìn thấy một vùng ánh sáng chói do toàn bộ năng lượng của ánh sáng được truyền thẳng vào mắt. Muốn tính màu specular của điểm ảnh ta phải xác định được mức độ ánh sáng phản chiếu tại điểm đó. Công thức tính vector phản chiếu (phản xạ) như sau:



Hình 28. Sự phản xạ của tia sáng trên bề mặt

$$R = 2(L \text{ dot } N)N - L$$

Trong đó:

L. Light vector

R. Reflection vector

N. Normal

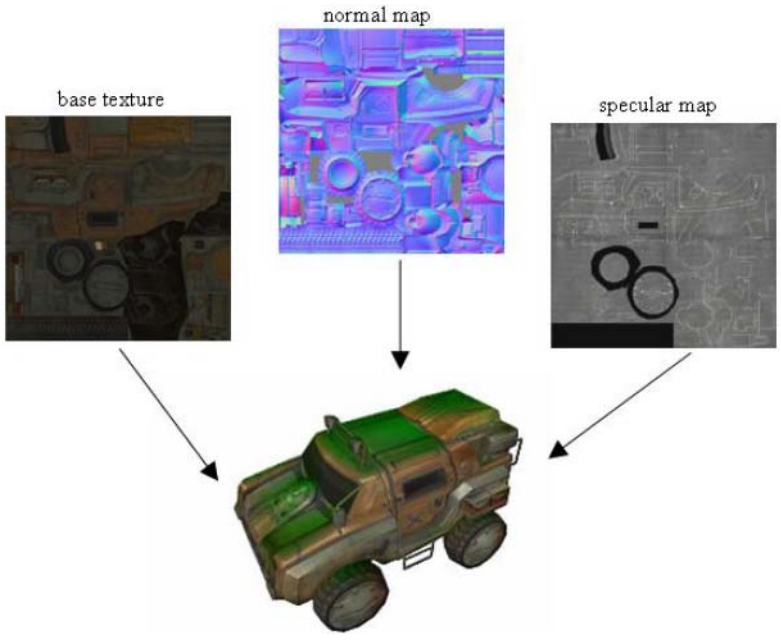
Mức độ phản chiếu của ánh sáng phụ thuộc rất nhiều vào chất liệu bề mặt của vật thể, các bề mặt nhẵn bóng có độ phản chiếu lớn trong khi các bề mặt gồ ghề lại có độ phản chiếu thấp. Để tránh công việc phải phân rã vật thể ra thành nhiều thành phần để dựng hình với các mức phản chiếu khác nhau người ta dùng specular map như một lookup table để xác định mức độ phản chiếu của ánh sánh trên từng điểm ảnh.

```
reflection vector = 2 * dotproduct (normal, light vector) * normal - light vector
specular factor = dotproduct (reflection vector, view vector)
specular color = (specular factor ^ specular constant) * specular lookup
```

Trong đó

- normal. Vector pháp tuyến (normal vector) tại điểm đó, normal vector có được do lấy mẫu từ normal map.
- light vector. Vector hướng ánh sáng trong không gian tiếp tuyến (tangent space), vector này được tính trong Vertex Shader và được truyền vào Pixel Shader để sử dụng.
- view vector. Vector tính từ mắt đến điểm nhìn (tọa độ trong không gian tiếp tuyến), vector này được tính trong Vertex Shader và truyền vào Pixel Shader để sử dụng specular constant. Hằng phản chiếu, giá trị càng lớn thì vùng phản chiếu càng nhỏ.
- specular lookup. Cho biết mức độ phản chiếu của điểm đó, giá trị này có được do lấy mẫu từ specular map. Sau khi thêm độ phản chiếu anh sáng vào xe trông như là được cấu tạo từ kim loại, do đó sự phản chiếu góp phần tăng đáng kể chất lượng đồ họa.

Hình 29. Tính độ phản chiếu trên từng điểm ảnh



Hình 30. Tóm tắt qui trình per-pixel lighting bằng hình vẽ

Ví dụ thể hiện kỹ thuật Bump mapping trên nền thư viện OpenSceneGraph

bumpmap.frag

```
uniform int       EnableNormalMapping;
uniform int       NumLights;
uniform sampler2D BaseMap;
uniform sampler2D NormalMap;

varying vec4      lightDir[ 3 ];    // the w component contains the
distance to light
varying vec3      viewDir;
varying vec3      normal;

// function for point lighting
// l_index:  light index
// l_dist:   light distance
// s_normal: surface normal for current fragment ( grabed from normal map )
// color:    resulting color
void pointlight( in int l_index, in float l_dist, in vec3 s_normal, inout
vec3 color )
{
    vec3 ldir = normalize( lightDir[ l_index ].xyz );
    float NdotL = max( dot( s_normal, ldir ), 0.0 );
    if ( NdotL > 0.0 )
    {
        // calculate the color contributions
        vec3 diffuse       = gl_FrontMaterial.diffuse.xyz * gl_LightSource[
l_index ].diffuse.xyz;
        vec3 ambient       = gl_FrontMaterial.ambient.xyz * gl_LightSource[
l_index ].ambient.xyz;
        vec3 ambientGlobal = gl_LightModel.ambient.xyz    *
gl_FrontMaterial.ambient.xyz;

        float att = 1.0 / ( gl_LightSource[ l_index ].constantAttenuation +
                            gl_LightSource[ l_index ].linearAttenuation *
l_dist +
                            gl_LightSource[ l_index ].quadraticAttenuation
* l_dist * l_dist
                          );

        // calculate the diffuse and ambient part
        color += att * ( diffuse * NdotL + ambient );

        float d = 2.0 * dot( ldir, s_normal );
        vec3  r = ldir - ( d * s_normal );
        float spec = pow( max( dot( r, viewDir ), 0.0 ),
gl_FrontMaterial.shininess );

        // calculate the specular part
        color += att * vec3( gl_FrontMaterial.specular * gl_LightSource[
l_index ].specular * spec );
    }
}

// function for spot lighting
// l_index:  light index
// l_dist:   light distance
// s_normal: surface normal for current fragment ( grabed from normal map )
// color:    resulting color
void spotlight( in int l_index, in float l_dist, in vec3 s_normal, inout
vec3 color )
{
    vec3 ldir = normalize( lightDir[ l_index ].xyz );
```

```
    float NdotL = max( dot( s_normal, ldir ), 0.0 );
    if ( NdotL > 0.0 )
    {
        float spotEffect = dot( normalize( gl_LightSource[ l_index
].spotDirection ), ldir );
        if ( spotEffect > gl_LightSource[ l_index ].spotCosCutoff )
        {
            spotEffect = pow( spotEffect, gl_LightSource[ l_index
].spotExponent );

            // calculate the color contributions
            vec3 diffuse       = gl_FrontMaterial.diffuse.xyz *
gl_LightSource[ l_index ].diffuse.xyz;
            vec3 ambient       = gl_FrontMaterial.ambient.xyz *
gl_LightSource[ l_index ].ambient.xyz;
            vec3 ambientGlobal = gl_LightModel.ambient.xyz    *
gl_FrontMaterial.ambient.xyz;

            float att = spotEffect / ( gl_LightSource[ l_index
].constantAttenuation +
                                       gl_LightSource[ l_index
].linearAttenuation * l_dist +
                                       gl_LightSource[ l_index
].quadraticAttenuation * l_dist * l_dist
                                     );

            // calculate the diffuse and ambient part
            color += att * ( diffuse * NdotL + ambient );

            float d = 2.0 * dot( ldir, s_normal );
            vec3  r = ldir - ( d * s_normal );
            float spec = pow( max( dot( r, viewDir ), 0.0 ),
gl_FrontMaterial.shininess );

            // calculate the specular part
            color += att * vec3( gl_FrontMaterial.specular *
gl_LightSource[ l_index ].specular * spec );
        }
    }
}

// function for directional lighting
// l_index:  light index
// s_normal: surface normal for current fragment ( grabed from normal map )
// color:    resulting color
void directionallight( in int l_index, in vec3 s_normal, inout vec3 color )
{
    vec3 ldir = normalize( lightDir[ l_index ].xyz );
    float NdotL = max( dot( s_normal, ldir ), 0.0 );
    if ( NdotL > 0.0 )
    {
        // calculate the color contributions
        vec3 diffuse       = gl_FrontMaterial.diffuse.xyz * gl_LightSource[
l_index ].diffuse.xyz;
        vec3 ambient       = gl_FrontMaterial.ambient.xyz * gl_LightSource[
l_index ].ambient.xyz;
        vec3 ambientGlobal = gl_LightModel.ambient.xyz    *
gl_FrontMaterial.ambient.xyz;

        // calculate the diffuse and ambient part
        color += ( diffuse * NdotL + ambient );

        float d = 2.0 * dot( ldir, s_normal );
        vec3  r = ldir - ( d * s_normal );
```

```
        float spec = pow( max( dot( r, viewDir ), 0.0 ),
gl_FrontMaterial.shininess );

        // calculate the specular part
        color += vec3( gl_FrontMaterial.specular * gl_LightSource[ l_index
].specular * spec );
    }
}

// main fragment program
void main ()
{
    // fetch the base map
    vec4 basecolor = texture2D( BaseMap, vec2( gl_TexCoord[ 0 ] ) );

    // get the normal vector
    vec3 norm;
    if ( EnableNormalMapping == 0 )
    {
        norm = normalize( normal );
    }
    else
    {
        // get normal from normal map
        norm   = vec3( texture2D( NormalMap, vec2( gl_TexCoord[ 1 ] ) ) );
        norm   = ( norm - 0.5 ) * 2.0;
        norm.z = -norm.z;
    }

    // calculate lighting for up to 3 light sources
    vec3 color     = vec3( 0.0, 0.0, 0.0 );
    if ( NumLights > 0 )
    {
        if ( gl_LightSource[ 0 ].position.w == 0.0 )
            directionallight( 0, norm, color );
        else if ( gl_LightSource[ 0 ].spotCutoff == 180.0 )
            pointlight( 0, lightDir[ 0 ].w, norm, color );
        else
            spotlight( 0, lightDir[ 0 ].w, norm, color );
    }

    if ( NumLights > 1 )
    {
        if ( gl_LightSource[ 1 ].position.w == 0.0 )
            directionallight( 1, norm, color );
        else if ( gl_LightSource[ 1 ].spotCutoff == 180.0 )
            pointlight( 1, lightDir[ 1 ].w, norm, color );
        else
            spotlight( 1, lightDir[ 1 ].w, norm, color );
    }

    if ( NumLights > 2 )
    {
        if ( gl_LightSource[ 2 ].position.w == 0.0 )
            directionallight( 2, norm, color );
        else if ( gl_LightSource[ 2 ].spotCutoff == 180.0 )
            pointlight( 2, lightDir[ 2 ].w, norm, color );
        else
            spotlight( 2, lightDir[ 2 ].w, norm, color );
    }

    // add the emissive part only once per fragment
    color += gl_FrontMaterial.emission.xyz;
```

```
    // calculate the final color
    color  = clamp( basecolor.xyz * color, 0.0, 1.0 );

    // consider the alpha value of base map for transparency
    gl_FragColor = vec4( color, basecolor.w );
}
```

bumpmap.vert

```
uniform int       EnableNormalMapping;
uniform int       NumLights;

varying vec4      lightDir[ 3 ];      // the w component contains the
distance to light
varying vec3      viewDir;
varying vec3      normal;

attribute vec4    Tangent;
attribute vec4    Binormal;

void prepareNormalMapping( in vec3 eyePos )
{
    // we pick up to first 3 light sources for dynamic lighting
    if ( NumLights > 0 )
    {
        // transform the input vectors
        vec3 t = normalize( gl_NormalMatrix * Tangent.xyz );
        vec3 b = normalize( gl_NormalMatrix * Binormal.xyz );
        vec3 n = normalize( gl_NormalMatrix * gl_Normal );

        // transform light positions into surface local coordinates
        // up to 3 light sources are considered
        vec3 v;
        vec3 lp;
        if ( NumLights > 0 )
        {
            lp = eyePos - gl_LightSource[ 0 ].position.xyz;
            v.x = dot( lp, t );
            v.y = dot( lp, b );
            v.z = dot( lp, n );
            lightDir[ 0 ] = vec4( normalize( v ), length( lp ) );
        }
        if ( NumLights > 1 )
        {
            lp = eyePos - gl_LightSource[ 1 ].position.xyz;
            v.x = dot( lp, t );
            v.y = dot( lp, b );
            v.z = dot( lp, n );
            lightDir[ 1 ] = vec4( normalize( v ), length( lp ) );
        }
        if ( NumLights > 2 )
        {
            lp = eyePos - gl_LightSource[ 2 ].position.xyz;
            v.x = dot( lp, t );
            v.y = dot( lp, b );
            v.z = dot( lp, n );
            lightDir[ 2 ] = vec4( normalize( v ), length( lp ) );
        }
        // calculate the view direction
        v.x = dot( eyePos, t );
        v.y = dot( eyePos, b );
        v.z = dot( eyePos, n );
        viewDir = -normalize( v );
    }
```

```
}

void prepareBaseMapping( in vec3 eyePos )
{
    // we pick up to first 3 light sources for dynamic lighting
    if ( NumLights > 0 )
    {
        // transform the normal vector
        normal = normalize( gl_NormalMatrix * gl_Normal );

        // transform light positions into surface local coordinates
        // up to 3 light sources are considered
        vec3 v;
        if ( NumLights > 0 )
        {
            vec3 lp = gl_LightSource[ 0 ].position.xyz - eyePos;
            lightDir[ 0 ] = vec4( normalize( lp ), length( lp ) );
        }
        if ( NumLights > 1 )
        {
            vec3 lp = gl_LightSource[ 1 ].position.xyz - eyePos;
            lightDir[ 1 ] = vec4( normalize( lp ), length( lp ) );
        }
        if ( NumLights > 2 )
        {
            vec3 lp = gl_LightSource[ 2 ].position.xyz - eyePos;
            lightDir[ 2 ] = vec4( normalize( lp ), length( lp ) );
        }
        // calculate the view direction
        viewDir = normalize( eyePos );
    }
}

void main()
{
    gl_Position = gl_ModelViewProjectionMatrix * gl_Vertex;
    vec3 eyePos = vec3( gl_ModelViewMatrix * gl_Vertex );
    gl_TexCoord[ 0 ] = gl_MultiTexCoord0;
    gl_TexCoord[ 1 ] = gl_MultiTexCoord1;
    gl_TexCoord[ 2 ] = gl_MultiTexCoord2;

    if ( EnableNormalMapping == 0 )
        prepareBaseMapping( eyePos );
    else
        prepareNormalMapping( eyePos );
}
```

vtBumpMapDrawable.h

```
#ifndef VT_BUMP_MAPPING_H
#define VT_BUMP_MAPPING_H 1

#include <osg/Group>
#include <osg/PositionAttitudeTransform>
#include <osg/Drawable>
#include <osg/Geode>
#include <osg/Geometry>
#include <osg/Image>
#include <osg/Texture2D>
#include <osg/PrimitiveSet>
#include <osg/Program>
#include <osgDB/ReadFile>
#include <osg/Uniform>
```

```
#include <osg/Vec3>
#include <osg/Vec4>

class vtBumpMapDrawable: public osg::Geode
{

public:
   vtBumpMapDrawable();
   ~vtBumpMapDrawable();
   vtBumpMapDrawable(const vtBumpMapDrawable& drawable,const osg::CopyOp&
copyop=osg::CopyOp::SHALLOW_COPY):
          osg::Geode(drawable,copyop)
              {}
      //##ModelId=423C65800117
      META_Node(osg, vtBumpMapDrawable);

   void init();

   void setUniforms(const osg::Vec3& pLightPos, const osg::Vec3& pEyePos);

};

#endif
```

vtBumpMapDrawable.cpp

```
#include "vtBumpmapDrawable.h"
#include "vtCubeVertices.h"

#include <osg/PolygonMode>

vtBumpMapDrawable::vtBumpMapDrawable()
{
      vtCubeVertices mData;

   osg::ref_ptr<osg::Geometry> _geometry = new osg::Geometry;

   //set our geometry variables

   //the actual verts
   _geometry->setVertexArray(mData.mVerts.get());
   _geometry->setNormalArray(mData.mNormals.get());

   //verts 2 and 3
   _geometry->setTexCoordArray(0, mData.mVerts2.get());
   _geometry->setTexCoordArray(1, mData.mVerts3.get());

   //tex coords 1-3

   _geometry->setTexCoordArray(2, mData.mTexCoords.get());
   _geometry->setTexCoordArray(3, mData.mTexCoords2.get());
   _geometry->setTexCoordArray(4, mData.mTexCoords3.get());

   //create indices
   osg::ref_ptr<osg::IntArray> indices =
      new osg::IntArray(36);

   for(int i = 0; i < 36; ++i)
   {
      (*indices)[i] = i;
   }

   _geometry->setVertexIndices(indices.get());
```

```
   _geometry->setNormalIndices(indices.get());

   _geometry->setTexCoordIndices(0, indices.get());
   _geometry->setTexCoordIndices(1, indices.get());
   _geometry->setTexCoordIndices(2, indices.get());
   _geometry->setTexCoordIndices(3, indices.get());
   _geometry->setTexCoordIndices(4, indices.get());

   _geometry->addPrimitiveSet( new
osg::DrawArrays(osg::PrimitiveSet::TRIANGLES, 0, 36) );

   _geometry->setNormalBinding(osg::Geometry::BIND_PER_VERTEX);

   addDrawable(_geometry.get());
   setName("BumpMapCube");



     //create state set
   osg::StateSet* ss = getOrCreateStateSet();
   //ss->setMode(osg::StateAttribute::CULLFACE, GL_FRONT);

   //set up textures
   osg::ref_ptr<osg::Image> img1 = osgDB::readImageFile("sheetmetal.tga"));
   osg::ref_ptr<osg::Image> img2 = osgDB::readImageFile("normal_map.tga"));


   osg::ref_ptr<osg::Texture2D> tex1 = new osg::Texture2D(img1.get());
   osg::ref_ptr<osg::Texture2D> tex2 = new osg::Texture2D(img2.get());

   tex1->setWrap(osg::Texture::WRAP_S, osg::Texture::REPEAT);
   tex1->setWrap(osg::Texture::WRAP_T, osg::Texture::REPEAT);

   tex2->setWrap(osg::Texture::WRAP_S, osg::Texture::REPEAT);
   tex2->setWrap(osg::Texture::WRAP_T, osg::Texture::REPEAT);

   ss->setTextureAttributeAndModes(0,tex1.get(),osg::StateAttribute::ON);
   ss->setTextureAttributeAndModes(1,tex2.get(),osg::StateAttribute::ON);

   ss->setTextureMode(0, GL_TEXTURE_2D, GL_TRUE);
   ss->setTextureMode(1, GL_TEXTURE_2D, GL_TRUE);


   //load the shader file
   osg::ref_ptr<osg::Program> _prog = new osg::Program;

   osg::ref_ptr<osg::Shader> bumpMapVert = new osg::Shader(
osg::Shader::VERTEX );
   osg::ref_ptr<osg::Shader> bumpMapFrag = new osg::Shader(
osg::Shader::FRAGMENT );

   _prog->addShader( bumpMapVert.get() );
   _prog->addShader( bumpMapFrag.get() );

   bumpMapVert->loadShaderSourceFromFile("shaders/bumpmap.vert");
   bumpMapFrag->loadShaderSourceFromFile("shaders/bumpmap.frag");

   osg::ref_ptr<osg::Uniform> _lightPos = new
osg::Uniform(osg::Uniform::FLOAT_VEC4, "lightPos");
   _lightPos->set(osg::Vec4(0.0f, 0.0f, 0.0f, 0.0f));
   ss->addUniform(_lightPos.get());
```

```
   osg::ref_ptr<osg::Uniform> _eyePos = new
osg::Uniform(osg::Uniform::FLOAT_VEC4, "eyePosition");
   _eyePos->set(osg::Vec4(0.0f, 0.0f, 0.0f, 0.0f));
   ss->addUniform(_eyePos.get());


   osg::ref_ptr<osg::Uniform> unTex1 = new
osg::Uniform(osg::Uniform::SAMPLER_2D, "texMap");
   unTex1->set(0);
   ss->addUniform(unTex1.get());


   osg::ref_ptr<osg::Uniform> unTex2 = new
osg::Uniform(osg::Uniform::SAMPLER_2D, "normalMap");
   unTex2->set(1);
   ss->addUniform(unTex2.get());

   ss->setAttributeAndModes( _prog.get(), osg::StateAttribute::ON );
}

vtBumpMapDrawable::~vtBumpMapDrawable()
{


}

void vtBumpMapDrawable::setUniforms(const osg::Vec3& pLightPos, const
osg::Vec3& pEyePos)
{
   osg::StateSet* ss = getOrCreateStateSet();
   ss->getUniform( "lightPos" )->set( osg::Vec4(pLightPos[0], pLightPos[1],
pLightPos[2], 0.0f));
   ss->getUniform( "eyePosition" )->set(osg::Vec4(pEyePos[0], pEyePos[2],
pEyePos[2], 0.0f));
}
```

vtCubeVertices.h

```
#ifndef VT_CUBE_VERTICES_H
#define VT_CUBE_VERTICES_H 1

#include <osg/ref_ptr>
#include <osg/Vec2>
#include <osg/Vec3>

typedef struct _CV_
{

public:

      _CV_();

       osg::ref_ptr<osg::Vec3Array> mVerts;
       osg::ref_ptr<osg::Vec3Array> mVerts2;
       osg::ref_ptr<osg::Vec3Array> mVerts3;

       osg::ref_ptr<osg::Vec3Array> mNormals;

       osg::ref_ptr<osg::Vec2Array> mTexCoords;
       osg::ref_ptr<osg::Vec2Array> mTexCoords2;
       osg::ref_ptr<osg::Vec2Array> mTexCoords3;

       float mTexRepeat;
       float mSize;
```

```
} CubeVertices;

////////////////////////////////////////////////////////////////////////
//Control Variables
////////////////////////////////////////////////////////////////////////

_CV_::_CV_()
{

      mTexRepeat = 1.0f;
      mSize = 1000.0f;

      mVerts = new osg::Vec3Array(36);
      mVerts2 = new osg::Vec3Array(36);
      mVerts3 = new osg::Vec3Array(36);
      mNormals = new osg::Vec3Array(36);
      mTexCoords = new osg::Vec2Array(36);
      mTexCoords2 = new osg::Vec2Array(36);
      mTexCoords3 = new osg::Vec2Array(36);

////////////////////////////////////////////////////////////////////////
//Verts 1
////////////////////////////////////////////////////////////////////////


//bottom
   (*mVerts)[0].set(mSize, mSize, -mSize);
   (*mVerts)[1].set(-mSize, mSize, -mSize);
   (*mVerts)[2].set(-mSize, -mSize, -mSize);

   (*mVerts)[3].set (mSize, -mSize, -mSize);
   (*mVerts)[4].set (mSize, mSize, -mSize);
   (*mVerts)[5].set (-mSize, -mSize, -mSize);

//top
   (*mVerts)[6].set (-mSize, mSize, mSize);
   (*mVerts)[7].set (mSize, mSize, mSize);
   (*mVerts)[8].set (-mSize, -mSize, mSize);

   (*mVerts)[9].set (mSize, mSize, mSize);
   (*mVerts)[10].set (mSize, -mSize, mSize);
   (*mVerts)[11].set (-mSize, -mSize, mSize);

//left
   (*mVerts)[12].set (-mSize, mSize, -mSize);
   (*mVerts)[13].set (-mSize, mSize, mSize);
   (*mVerts)[14].set (-mSize, -mSize, mSize);

   (*mVerts)[15].set (-mSize, -mSize, -mSize);
   (*mVerts)[16].set (-mSize, mSize, -mSize);
   (*mVerts)[17].set (-mSize, -mSize, mSize);

//right
   (*mVerts)[18].set (mSize, mSize, mSize);
   (*mVerts)[19].set (mSize, mSize, -mSize);
   (*mVerts)[20].set (mSize, -mSize, -mSize);

   (*mVerts)[21].set (mSize, -mSize, mSize);
   (*mVerts)[22].set (mSize, mSize, mSize);
   (*mVerts)[23].set (mSize, -mSize, -mSize);

//front
   (*mVerts)[24].set (mSize, mSize, mSize);
```

```
   (*mVerts)[25].set (-mSize, mSize, mSize);
   (*mVerts)[26].set (-mSize, mSize, -mSize);

   (*mVerts)[27].set (mSize, mSize, -mSize);
   (*mVerts)[28].set (mSize, mSize, mSize);
   (*mVerts)[29].set (-mSize, mSize, -mSize);

//back
   (*mVerts)[30].set (mSize, -mSize, -mSize);
   (*mVerts)[31].set (-mSize, -mSize, -mSize);
   (*mVerts)[32].set (-mSize, -mSize, mSize);

   (*mVerts)[33].set (mSize, -mSize, mSize);
   (*mVerts)[34].set (mSize, -mSize, -mSize);
   (*mVerts)[35].set (-mSize, -mSize, mSize);

////////////////////////////////////////////////////////////////////////
//Verts 2
////////////////////////////////////////////////////////////////////////

//bottom
   (*mVerts2)[0].set (-mSize, mSize, -mSize);
   (*mVerts2)[1].set (-mSize, -mSize, -mSize);
   (*mVerts2)[2].set (mSize, mSize, -mSize);

   (*mVerts2)[3].set (mSize, mSize, -mSize);
   (*mVerts2)[4].set (-mSize, -mSize, -mSize);
   (*mVerts2)[5].set (mSize, -mSize, -mSize);

//top
   (*mVerts2)[6].set (mSize, mSize, mSize);
   (*mVerts2)[7].set (-mSize, -mSize, mSize);
   (*mVerts2)[8].set (-mSize, mSize, mSize);

   (*mVerts2)[9].set (mSize, -mSize, mSize);
   (*mVerts2)[10].set (-mSize, -mSize, mSize);
   (*mVerts2)[11].set (mSize, mSize, mSize);

//left
   (*mVerts2)[12].set (-mSize, mSize, mSize);
   (*mVerts2)[13].set (-mSize, -mSize, mSize);
   (*mVerts2)[14].set (-mSize, mSize, -mSize);

   (*mVerts2)[15].set (-mSize, mSize, -mSize);
   (*mVerts2)[16].set (-mSize, -mSize, mSize);
   (*mVerts2)[17].set (-mSize, -mSize, -mSize);


//right
   (*mVerts2)[18].set (mSize, mSize, -mSize);
   (*mVerts2)[19].set (mSize, -mSize, -mSize);
   (*mVerts2)[20].set (mSize, mSize, mSize);

   (*mVerts2)[21].set (mSize, mSize, mSize);
   (*mVerts2)[22].set (mSize, -mSize, -mSize);
   (*mVerts2)[23].set (mSize, -mSize, mSize);

//front
   (*mVerts2)[24].set (-mSize, mSize, mSize);
   (*mVerts2)[25].set (-mSize, mSize, -mSize);
   (*mVerts2)[26].set (mSize, mSize, mSize);

   (*mVerts2)[27].set (mSize, mSize, mSize);
   (*mVerts2)[28].set (-mSize, mSize, -mSize);
```

```
   (*mVerts2)[29].set (mSize, mSize, -mSize);

//back
   (*mVerts2)[30].set (-mSize, -mSize, -mSize);
   (*mVerts2)[31].set (-mSize, -mSize, mSize);
   (*mVerts2)[32].set (mSize, -mSize, -mSize);

   (*mVerts2)[33].set (mSize, -mSize, -mSize);
   (*mVerts2)[34].set (-mSize, -mSize, mSize);
   (*mVerts2)[35].set (mSize, -mSize, mSize);

////////////////////////////////////////////////////////////////////////
//Verts 3
////////////////////////////////////////////////////////////////////////

//bottom
   (*mVerts3)[0].set (-mSize, -mSize, -mSize);
   (*mVerts3)[1].set (mSize, mSize, -mSize);
   (*mVerts3)[2].set (-mSize, mSize, -mSize);

   (*mVerts3)[3].set (-mSize, -mSize, -mSize);
   (*mVerts3)[4].set (mSize, -mSize, -mSize);
   (*mVerts3)[5].set (mSize, mSize, -mSize);

//top
   (*mVerts3)[6].set (-mSize, -mSize, mSize);
   (*mVerts3)[7].set (-mSize, mSize, mSize);
   (*mVerts3)[8].set (mSize, mSize, mSize);

   (*mVerts3)[9].set (-mSize, -mSize, mSize);
   (*mVerts3)[10].set (mSize, mSize, mSize);
   (*mVerts3)[11].set (mSize, -mSize, mSize);

//left
   (*mVerts3)[12].set (-mSize, -mSize, mSize);
   (*mVerts3)[13].set (-mSize, mSize, -mSize);
   (*mVerts3)[14].set (-mSize, mSize, mSize);

   (*mVerts3)[15].set (-mSize, -mSize, mSize    );
   (*mVerts3)[16].set (-mSize, -mSize, -mSize);
   (*mVerts3)[17].set (-mSize, mSize, -mSize);

//right
   (*mVerts3)[18].set (mSize, -mSize, -mSize);
   (*mVerts3)[19].set (mSize, mSize, mSize);
   (*mVerts3)[20].set (mSize, mSize, -mSize);

   (*mVerts3)[21].set (mSize, -mSize, -mSize);
   (*mVerts3)[22].set (mSize, -mSize, mSize);
   (*mVerts3)[23].set (mSize, mSize, mSize);

//front
   (*mVerts3)[24].set (-mSize, mSize, -mSize);
   (*mVerts3)[25].set (mSize, mSize, mSize);
   (*mVerts3)[26].set (-mSize, mSize, mSize);

   (*mVerts3)[27].set (-mSize, mSize, -mSize);
   (*mVerts3)[28].set (mSize, mSize, -mSize );
   (*mVerts3)[29].set (mSize, mSize, mSize);

//back
   (*mVerts3)[30].set (-mSize, -mSize, mSize);
   (*mVerts3)[31].set (mSize, -mSize, -mSize);
   (*mVerts3)[32].set (-mSize, -mSize, -mSize);
```

```
   (*mVerts3)[33].set (-mSize, -mSize, mSize);
   (*mVerts3)[34].set (mSize, -mSize, mSize);
   (*mVerts3)[35].set (mSize, -mSize, -mSize);

////////////////////////////////////////////////////////////////////////
//Normals
////////////////////////////////////////////////////////////////////////

//bottom
   (*mNormals)[0].set (0.0f, 0.0f, 1.0f);
   (*mNormals)[1].set (0.0f, 0.0f, 1.0f);
   (*mNormals)[2].set (0.0f, 0.0f, 1.0f);

   (*mNormals)[3].set (0.0f, 0.0f, 1.0f);
   (*mNormals)[4].set (0.0f, 0.0f, 1.0f);
   (*mNormals)[5].set (0.0f, 0.0f, 1.0f);

//top
   (*mNormals)[6].set (0.0f, 0.0f, -1.0f);
   (*mNormals)[7].set (0.0f, 0.0f, -1.0f);
   (*mNormals)[8].set (0.0f, 0.0f, -1.0f);

   (*mNormals)[9].set (0.0f, 0.0f, -1.0f);
   (*mNormals)[10].set (0.0f, 0.0f, -1.0f);
   (*mNormals)[11].set (0.0f, 0.0f, -1.0f);

//left
   (*mNormals)[12].set (1.0f, 0.0f, 0.0f);
   (*mNormals)[13].set (1.0f, 0.0f, 0.0f);
   (*mNormals)[14].set (1.0f, 0.0f, 0.0f);

   (*mNormals)[15].set (1.0f, 0.0f, 0.0f);
   (*mNormals)[16].set (1.0f, 0.0f, 0.0f);
   (*mNormals)[17].set (1.0f, 0.0f, 0.0f);

//right
   (*mNormals)[18].set (-1.0f, 0.0f, 0.0f);
   (*mNormals)[19].set (-1.0f, 0.0f, 0.0f);
   (*mNormals)[20].set (-1.0f, 0.0f, 0.0f);

   (*mNormals)[21].set (-1.0f, 0.0f, 0.0f);
   (*mNormals)[22].set (-1.0f, 0.0f, 0.0f);
   (*mNormals)[23].set (-1.0f, 0.0f, 0.0f);

//front
   (*mNormals)[24].set (0.0f, -1.0f, 0.0f);
   (*mNormals)[25].set (0.0f, -1.0f, 0.0f);
   (*mNormals)[26].set (0.0f, -1.0f, 0.0f);

   (*mNormals)[27].set (0.0f, -1.0f, 0.0f);
   (*mNormals)[28].set (0.0f, -1.0f, 0.0f);
   (*mNormals)[29].set (0.0f, -1.0f, 0.0f);

//back
   (*mNormals)[30].set (0.0f, 1.0f, 0.0f);
   (*mNormals)[31].set (0.0f, 1.0f, 0.0f);
   (*mNormals)[32].set (0.0f, 1.0f, 0.0f);

   (*mNormals)[33].set (0.0f, 1.0f, 0.0f);
   (*mNormals)[34].set (0.0f, 1.0f, 0.0f);
   (*mNormals)[35].set (0.0f, 1.0f, 0.0f);

////////////////////////////////////////////////////////////////////////
```

```
// Tex Coords1
//////////////////////////////////////////////////////////////////////////

//bottom
   (*mTexCoords)[0].set (mTexRepeat, mTexRepeat);
   (*mTexCoords)[1].set (-mTexRepeat, mTexRepeat);
   (*mTexCoords)[2].set (-mTexRepeat, -mTexRepeat);

   (*mTexCoords)[3].set (mTexRepeat, -mTexRepeat);
   (*mTexCoords)[4].set (mTexRepeat, mTexRepeat);
   (*mTexCoords)[5].set (-mTexRepeat, -mTexRepeat);

//top

   (*mTexCoords)[6].set (-mTexRepeat, -mTexRepeat);
   (*mTexCoords)[7].set (mTexRepeat, -mTexRepeat);
   (*mTexCoords)[8].set (-mTexRepeat, mTexRepeat);

   (*mTexCoords)[9].set (mTexRepeat, -mTexRepeat);
   (*mTexCoords)[10].set (mTexRepeat, mTexRepeat);
   (*mTexCoords)[11].set (-mTexRepeat, mTexRepeat);

//left
   (*mTexCoords)[12].set (mTexRepeat, -mTexRepeat);
   (*mTexCoords)[13].set (mTexRepeat, mTexRepeat);
   (*mTexCoords)[14].set (-mTexRepeat, mTexRepeat);

   (*mTexCoords)[15].set (-mTexRepeat, -mTexRepeat);
   (*mTexCoords)[16].set (mTexRepeat, -mTexRepeat);
   (*mTexCoords)[17].set (-mTexRepeat, mTexRepeat);

//right

   (*mTexCoords)[18].set (-mTexRepeat, mTexRepeat);
   (*mTexCoords)[19].set (-mTexRepeat, -mTexRepeat);
   (*mTexCoords)[20].set (mTexRepeat, -mTexRepeat);

   (*mTexCoords)[21].set (mTexRepeat, mTexRepeat);
   (*mTexCoords)[22].set (-mTexRepeat, mTexRepeat);
   (*mTexCoords)[23].set (mTexRepeat, -mTexRepeat);

//front
   (*mTexCoords)[24].set (mTexRepeat, mTexRepeat);
   (*mTexCoords)[25].set (-mTexRepeat, mTexRepeat);
   (*mTexCoords)[26].set (-mTexRepeat, -mTexRepeat);

   (*mTexCoords)[27].set (mTexRepeat, -mTexRepeat);
   (*mTexCoords)[28].set (mTexRepeat, mTexRepeat);
   (*mTexCoords)[29].set (-mTexRepeat, -mTexRepeat);

//back
   (*mTexCoords)[30].set (-mTexRepeat, -mTexRepeat);
   (*mTexCoords)[31].set (mTexRepeat, -mTexRepeat);
   (*mTexCoords)[32].set (mTexRepeat, mTexRepeat);

   (*mTexCoords)[33].set (-mTexRepeat, mTexRepeat);
   (*mTexCoords)[34].set (-mTexRepeat, -mTexRepeat);
   (*mTexCoords)[35].set (mTexRepeat, mTexRepeat);

//////////////////////////////////////////////////////////////////////////
// TexCoords 2
//////////////////////////////////////////////////////////////////////////
//bottom
```

```
   (*mTexCoords2)[0].set (-mTexRepeat, mTexRepeat);
   (*mTexCoords2)[1].set (-mTexRepeat, -mTexRepeat);
   (*mTexCoords2)[2].set (mTexRepeat, mTexRepeat);

   (*mTexCoords2)[3].set (mTexRepeat, mTexRepeat);
   (*mTexCoords2)[4].set (-mTexRepeat, -mTexRepeat);
   (*mTexCoords2)[5].set (mTexRepeat, -mTexRepeat);


//top

   (*mTexCoords2)[6].set (mTexRepeat, -mTexRepeat);
   (*mTexCoords2)[7].set (-mTexRepeat, mTexRepeat);
   (*mTexCoords2)[8].set (-mTexRepeat, -mTexRepeat);

   (*mTexCoords2)[9].set (mTexRepeat, mTexRepeat);
   (*mTexCoords2)[10].set (-mTexRepeat, mTexRepeat);
   (*mTexCoords2)[11].set (mTexRepeat, -mTexRepeat);

//left

   (*mTexCoords2)[12].set (mTexRepeat, mTexRepeat);
   (*mTexCoords2)[13].set (-mTexRepeat, mTexRepeat);
   (*mTexCoords2)[14].set (mTexRepeat, -mTexRepeat);

   (*mTexCoords2)[15].set (mTexRepeat, -mTexRepeat);
   (*mTexCoords2)[16].set (-mTexRepeat, mTexRepeat);
   (*mTexCoords2)[17].set (-mTexRepeat, -mTexRepeat);


//right
   (*mTexCoords2)[18].set (-mTexRepeat, -mTexRepeat);
   (*mTexCoords2)[19].set (mTexRepeat, -mTexRepeat);
   (*mTexCoords2)[20].set (-mTexRepeat, mTexRepeat);

   (*mTexCoords2)[21].set (-mTexRepeat, mTexRepeat);
   (*mTexCoords2)[22].set (mTexRepeat, -mTexRepeat);
   (*mTexCoords2)[23].set (mTexRepeat, mTexRepeat);

//front
   (*mTexCoords2)[24].set (-mTexRepeat, mTexRepeat);
   (*mTexCoords2)[25].set (-mTexRepeat, -mTexRepeat);
   (*mTexCoords2)[26].set (mTexRepeat, mTexRepeat);

   (*mTexCoords2)[27].set (mTexRepeat, mTexRepeat);
   (*mTexCoords2)[28].set (-mTexRepeat, -mTexRepeat);
   (*mTexCoords2)[29].set (mTexRepeat, -mTexRepeat);

//back
   (*mTexCoords2)[30].set (mTexRepeat, -mTexRepeat);
   (*mTexCoords2)[31].set (mTexRepeat, mTexRepeat);
   (*mTexCoords2)[32].set (-mTexRepeat, -mTexRepeat);

   (*mTexCoords2)[33].set (-mTexRepeat, -mTexRepeat);
   (*mTexCoords2)[34].set (mTexRepeat, mTexRepeat);
   (*mTexCoords2)[35].set (-mTexRepeat, mTexRepeat);

////////////////////////////////////////////////////////////////////////
// Tex Coords 3
////////////////////////////////////////////////////////////////////////

//bottom

   (*mTexCoords3)[0].set (-mTexRepeat, -mTexRepeat);
```

```
   (*mTexCoords3)[1].set (mTexRepeat, mTexRepeat);
   (*mTexCoords3)[2].set (-mTexRepeat, mTexRepeat);

   (*mTexCoords3)[3].set (-mTexRepeat, -mTexRepeat);
   (*mTexCoords3)[4].set (mTexRepeat, -mTexRepeat);
   (*mTexCoords3)[5].set (mTexRepeat, mTexRepeat);


//top

   (*mTexCoords3)[6].set (-mTexRepeat, mTexRepeat);
   (*mTexCoords3)[7].set (-mTexRepeat, -mTexRepeat);
   (*mTexCoords3)[8].set (mTexRepeat, -mTexRepeat);

   (*mTexCoords3)[9].set (-mTexRepeat, mTexRepeat);
   (*mTexCoords3)[10].set (mTexRepeat, -mTexRepeat);
   (*mTexCoords3)[11].set (mTexRepeat, mTexRepeat);


//left
   (*mTexCoords3)[12].set (-mTexRepeat, mTexRepeat);
   (*mTexCoords3)[13].set (mTexRepeat, -mTexRepeat);
   (*mTexCoords3)[14].set (mTexRepeat, mTexRepeat);

   (*mTexCoords3)[15].set (-mTexRepeat, mTexRepeat);
   (*mTexCoords3)[16].set (-mTexRepeat, -mTexRepeat);
   (*mTexCoords3)[17].set (mTexRepeat, -mTexRepeat);


//right
   (*mTexCoords3)[18].set (mTexRepeat, -mTexRepeat);
   (*mTexCoords3)[19].set (-mTexRepeat, mTexRepeat);
   (*mTexCoords3)[20].set (-mTexRepeat, -mTexRepeat);

   (*mTexCoords3)[21].set (mTexRepeat, -mTexRepeat);
   (*mTexCoords3)[22].set (mTexRepeat, mTexRepeat);
   (*mTexCoords3)[23].set (-mTexRepeat, mTexRepeat);

//front
   (*mTexCoords3)[24].set (-mTexRepeat, -mTexRepeat);
   (*mTexCoords3)[25].set (mTexRepeat, mTexRepeat);
   (*mTexCoords3)[26].set (-mTexRepeat, mTexRepeat);

   (*mTexCoords3)[27].set (-mTexRepeat, -mTexRepeat);
   (*mTexCoords3)[28].set (mTexRepeat, -mTexRepeat);
   (*mTexCoords3)[29].set (mTexRepeat, mTexRepeat);

//back
   (*mTexCoords3)[30].set (mTexRepeat, mTexRepeat);
   (*mTexCoords3)[31].set (-mTexRepeat, -mTexRepeat);
   (*mTexCoords3)[32].set (mTexRepeat, -mTexRepeat);

   (*mTexCoords3)[33].set (mTexRepeat, mTexRepeat);
   (*mTexCoords3)[34].set (-mTexRepeat, mTexRepeat);
   (*mTexCoords3)[35].set (-mTexRepeat, -mTexRepeat);

}

#endif
```

Tài liệu tham khảo

- Simulation Of Wrinkled Surfaces, Jim Blinn (not freely available online to my knowledge)
- Efficient Bump Mapping Hardware, Mark Peercy, John Airey and Brian Cabral
- Bump Mapping, Anders Hast
- Interactive Horizon Mapping, Peter-Pike Sloan and Michael F. Cohen
- Parallax Mapping with Offset Limiting: A Per-Pixel Approximation of Uneven Surfaces, Terry Welsh
- Preserving Attribute Values On Simplified Meshes By Resampling Detail Textures, P. Cignoni, C. Montani, C. Rocchini, R. Scopigno and M. Tarini
- All The Polygons You Can Eat: Without The Fat, Doug Rogers (includes some good information on detail preservation)
- Tangent Space Calculation, Eric Lengyel (example code for calculating tangent frames)

Công cụ

- ATI developer tools
- NVidia texture tools
- NVidia Melody
- Open RenderBump

Một số demo bằng ngôn ngữ Delphi:

- Emboss bump mapping.
- EMBM for fake refraction (NVidia only).
- The "*Hello world*" of normal mapping.
- Per-pixel lighting using normal maps. Different code paths for different hardware generations (NVidia only).
- Portal renderer with shadow mapping and per-pixel lighting using normal maps.
- Horizon mapping.
- A combination of horizon mapping, parallax mapping and z-correct bump mapping.
- A simple mesh-based normal map generator for generating tiled normal maps.