Bump Mapping

CSU0021: Computer Graphics

3D Model for a Delicate Object

- If you want to render a wall like this figure, what is the 3D model you need?
  - A color texture
  - A very complicated 3D wall model because the wall surface is undulating
    - You need a lot of small triangles to describe this 3D model

A Cheaper Way to Describe a Delicate 3D Model

- Why can we perceive the “3D” of the wall?
  - Illumination
  - key of the illumination: the normal vector

- Can the users perceive this complicated “3D” on a simple flat quad?
  - Yes, if we well control the normal vectors (do not simply use the normal vector which is perpendicular to the quad)

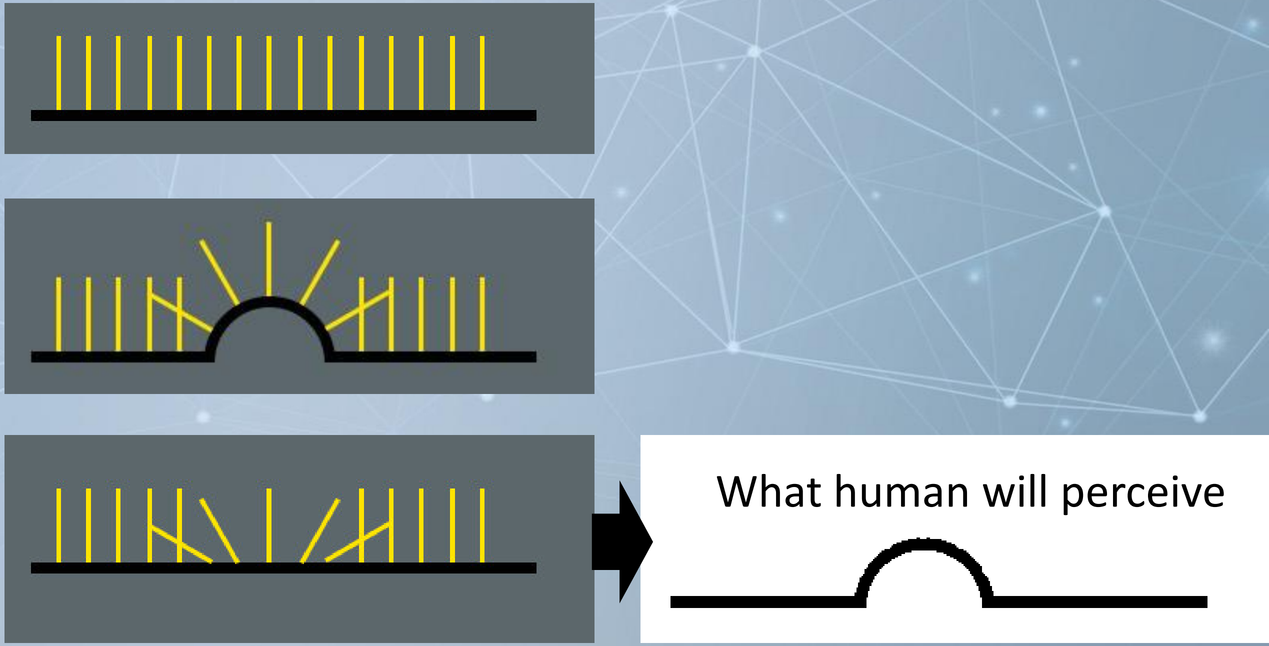
Bump Mapping

- Black: surface, yellow: normal vectors

A low poly surface with its normal vectors

A high poly surface with its normal vectors

A low poly surface with (fake) high poly surface's normal vectors

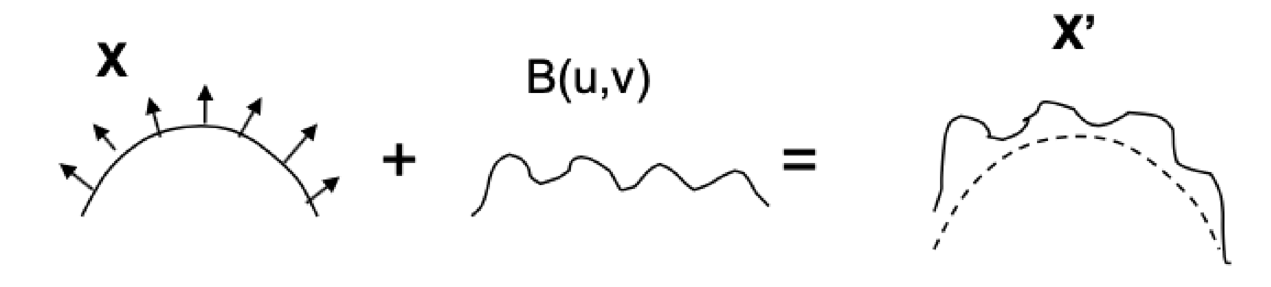


Bump Mapping

This is just a 3D object

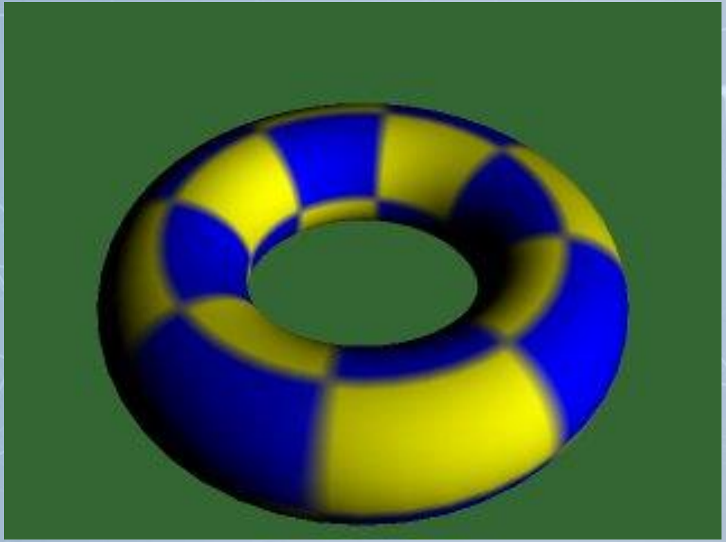
This is the **fake normals**

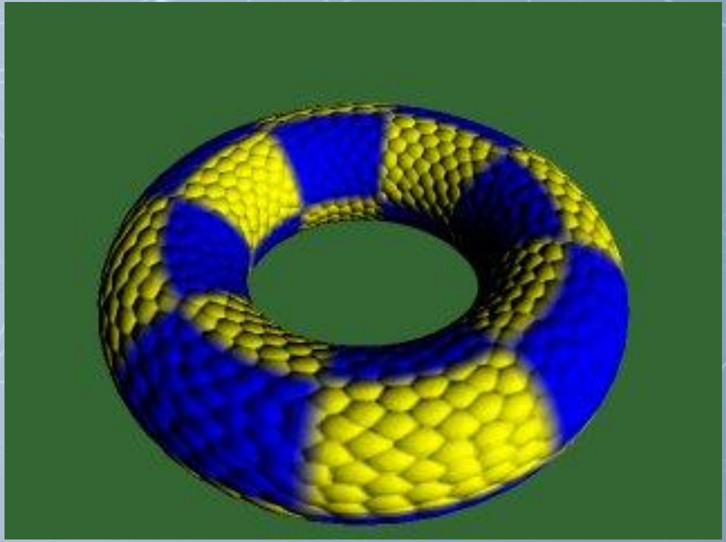
Render the object using the fake normals to have the illumination



How can we have the **fake normals**?

Bump Mapping Example





Bump Mapping Categories

Bump Map

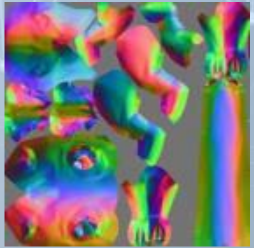
Displacement Map

Normal Map

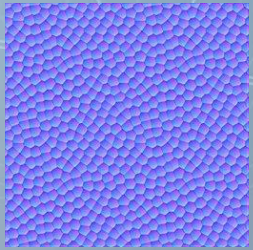
Scalar

Vectorial

Object Space

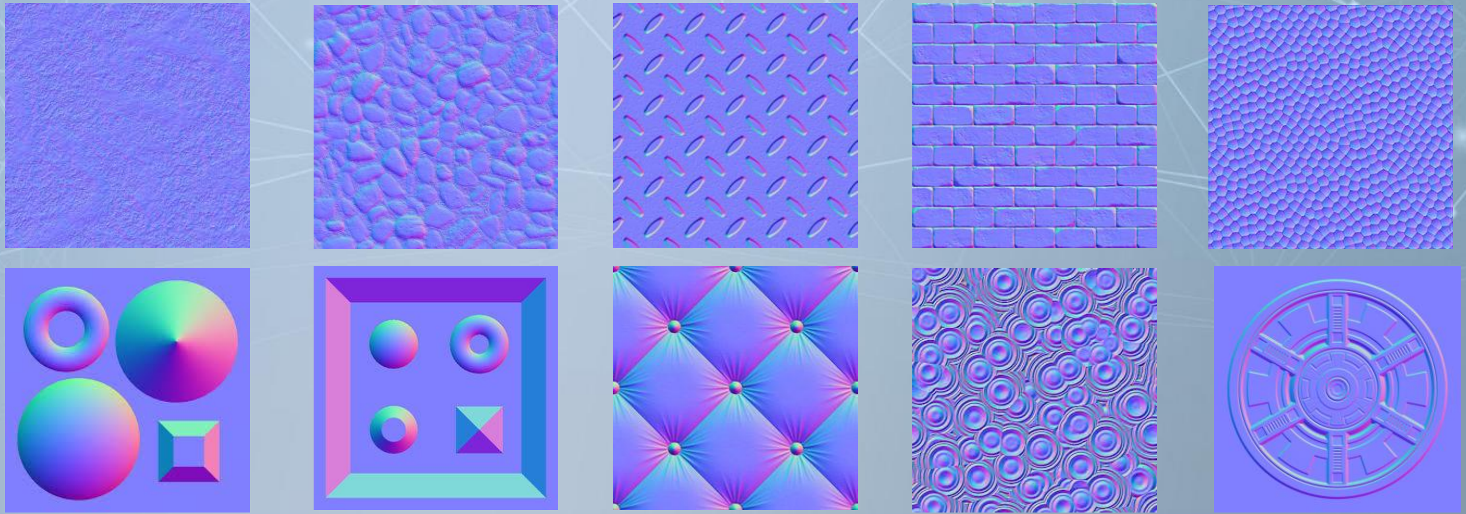


Tangent Space



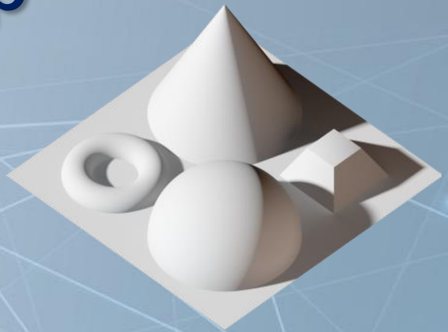
Normal Map

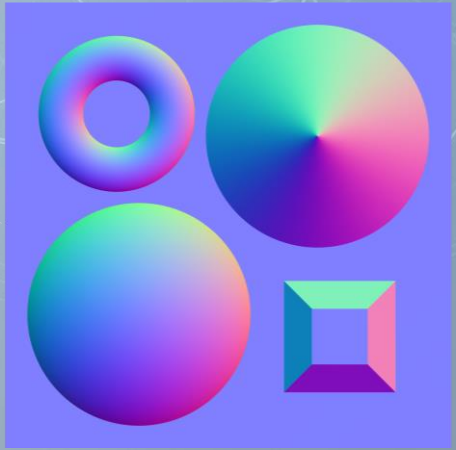
- We can store the fake normal vectors in an image and we call it normal map
- Each pixel (rgb) represents a normal vector (xyz)
  - Range of RGB: 0 ~ 1
  - Range of xyz of normal vector: -1 ~ +1
- **Remember to rescale the data range before you use a rgb in normal map as a normal vector**



How to Make a Normal Map

- (We will not create our own normal map in this course. You can search and download on the internet)
- The idea of one way to create a normal map
  - Create a real 3D object
  - Define a surface **S** (z=0) (and S will be the normal map plane)
  - For a point **p** on the surface
    - Calculate normal vectors of p
    - p’ is the projection point of p on S
    - Store the normal vector of p at p’
      - Of course, we have to rescale the normal vector range (-1 ~ +1) to rgb range (0 ~ 1) because we will store this normal map as an image

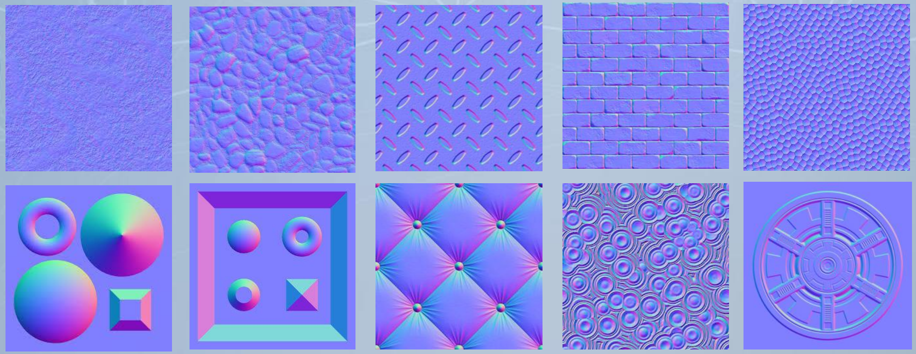


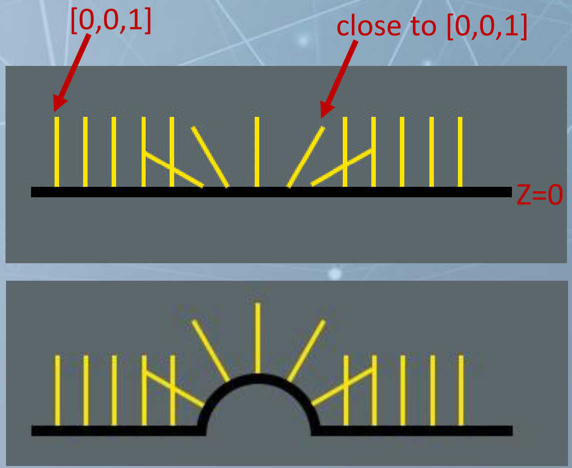


Why Normal Maps Always Looks Purple Blue?

- Short answer is: the projection plane is z plane, so most of the normal vectors are close to [0,0,1]
  - After rescaling it to rgb will be close to [0.5, 0.5, 1] (purple blue)

- Better answer: normal vectors in normal images are defined in tangent space (introduce later)





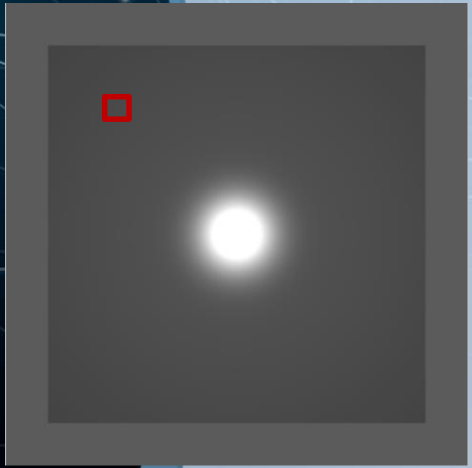
How to Use the Normal Map

- **When render a fragment on the plane (quad)**
- Using the texture coordinate of the fragment to look up the rgb in the normal map
- Rescale the rgb to be a vector range: vector = (rgb* 2 - 1)
- Use the vector to calculate the illumination of the fragment

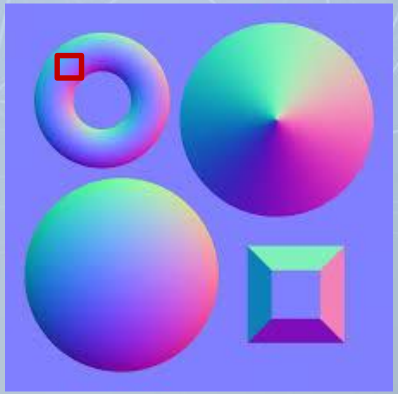
A quad

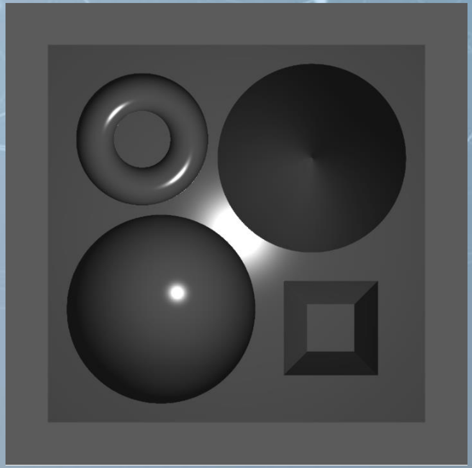
How to Use the Normal Map

- When render a fragment on the plane (quad)
- **Using the texture coordinate of the fragment to look up the rgb in the normal map**
- Rescale the rgb to be a vector range: vector = (rgb* 2 - 1)
- Use the vector to calculate the illumination of the fragment



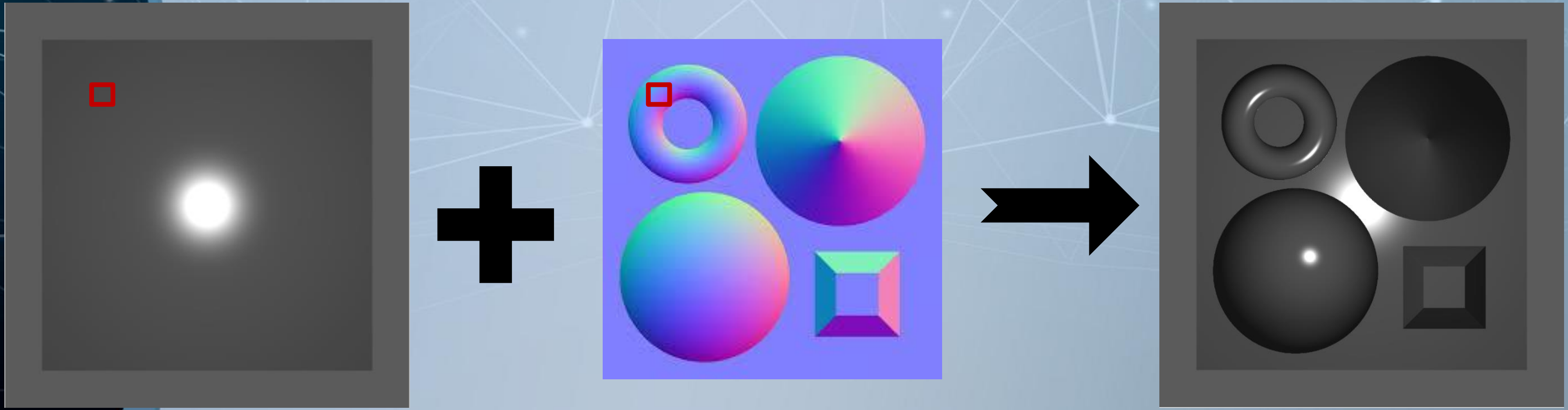
A quad





How to Use the Normal Map

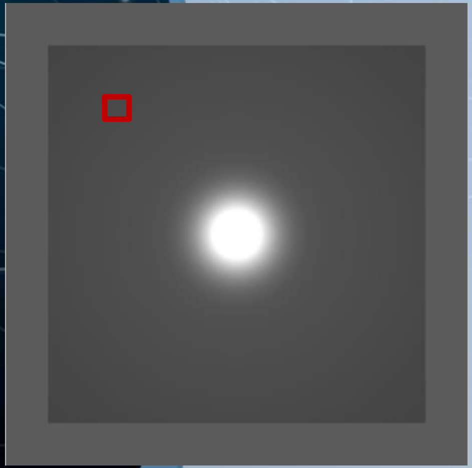
- When render a fragment on the plane (quad)
- Using the texture coordinate of the fragment to look up the rgb in the normal map
- **Rescale the rgb to be a vector range: vector = (rgb* 2 - 1)**
- Use the vector to calculate the illumination of the fragment



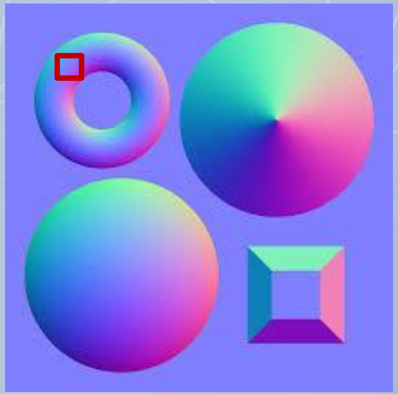
A quad

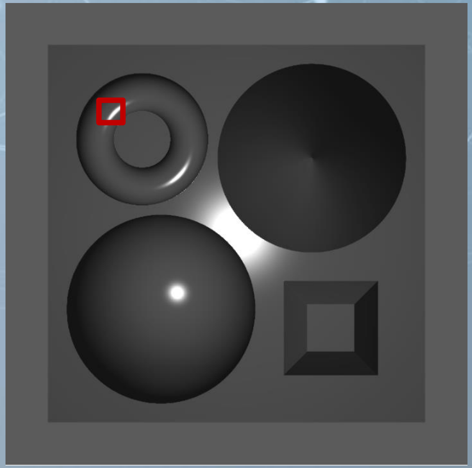
How to Use the Normal Map

- When render a fragment on the plane (quad)
- Using the texture coordinate of the fragment to look up the rgb in the normal map
- Rescale the rgb to be a vector range: vector = (rgb* 2 - 1)
- **Use the vector to calculate the illumination of the fragment**



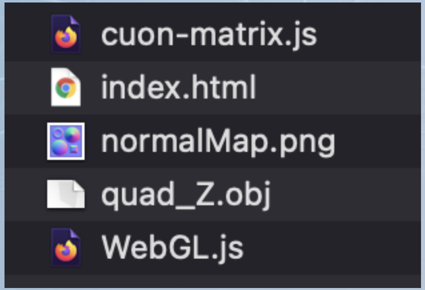
A quad

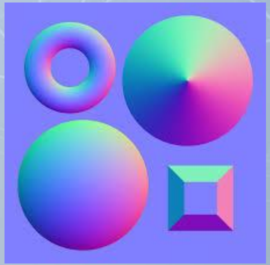


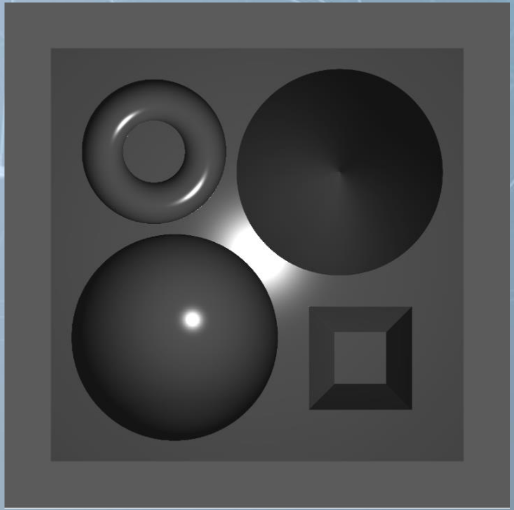


Example (Ex13-1)

- Bump mapping on a quad
- We are going to use the texture coordinate to look up vectors on the normal map to calculate the illumination
- I set the light source and the camera at the same location. It is easier to check the correctness of the bump mapping implementation
- Files







Example (Ex13-1)

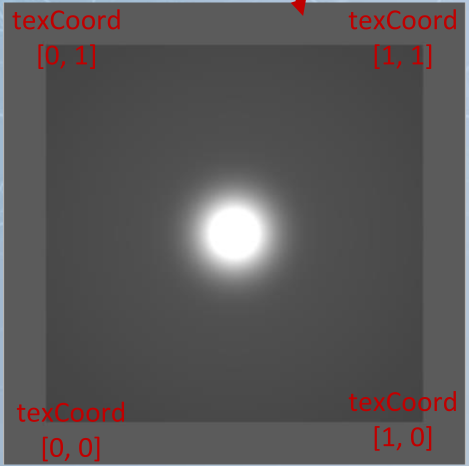
- quad_z.obj is a quad with z = 0
  - So, it's true normal vector are [0, 0, 1] (but, we will not use its true normal vector here)

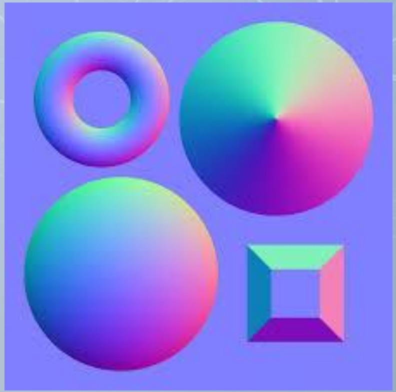
```
v -1 -1 0
v 1 -1 0
v -1 1 0
v 1 1 0
vn 0 0 1
vn 0 0 1
vn 0 0 1
vn 0 0 1
vt 0 0
vt 1 0
vt 0 1
vt 1 1
f 1/1/1 4/4/4 3/3/3
f 1/1/1 2/2/2 4/4/4
```

Example (Ex13-1)

- quad_z.obj is a quad with z = 0
  - So, it‘s true normal vector are [0, 0, 1] (but, we will not use its true normal vector here)
- The texture coordinate setting of quad in this obj file is

```
v -1 -1 0
v 1 -1 0
v -1 1 0
v 1 1 0
vn 0 0 1
vn 0 0 1
vn 0 0 1
vn 0 0 1
vt 0 0
vt 1 0
vt 0 1
vt 1 1
f 1/1/1 4/4/4 3/3/3
f 1/1/1 2/2/2 4/4/4
```

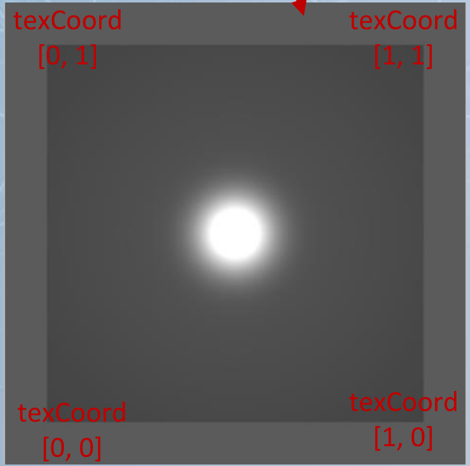


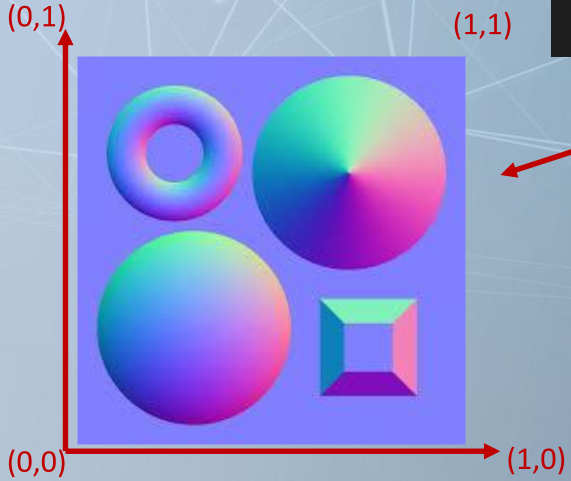


Example (Ex13-1)

- quad_z.obj is a quad with z = 0
  - So, it's true normal vector are [0, 0, 1] (but, we will not use its true normal vector here)
- The texture coordinate setting of quad in this obj file is

```
v -1 -1 0
v 1 -1 0
v -1 1 0
v 1 1 0
vn 0 0 1
vn 0 0 1
vn 0 0 1
vn 0 0 1
vt 0 0
vt 1 0
vt 0 1
vt 1 1
f 1/1/1 4/4/4 3/3/3
f 1/1/1 2/2/2 4/4/4
```



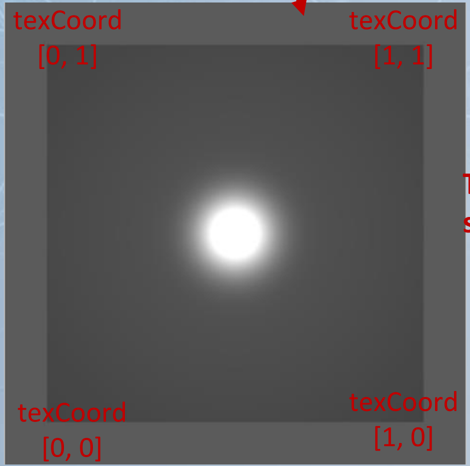


If we use this image as a texture

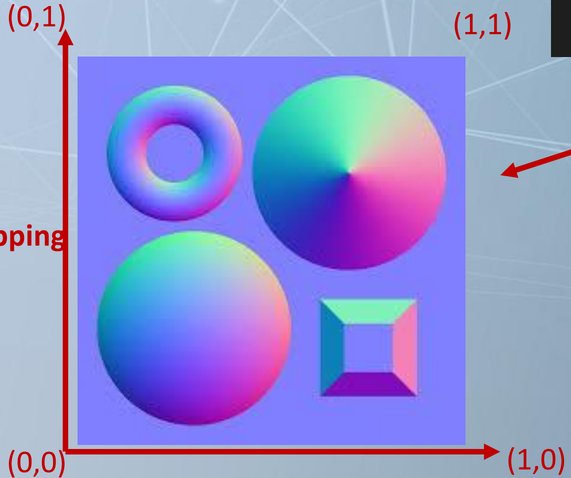
Example (Ex13-1)

- quad_z.obj is a quad with z = 0
  - So, it's true normal vector are [0, 0, 1] (but, we will not use its true normal vector here)
- The texture coordinate setting of quad in this obj file is

```
v -1 -1 0
v 1 -1 0
v -1 1 0
v 1 1 0
vn 0 0 1
vn 0 0 1
vn 0 0 1
vn 0 0 1
vt 0 0
vt 1 0
vt 0 1
vt 1 1
f 1/1/1 4/4/4 3/3/3
f 1/1/1 2/2/2 4/4/4
```



They are a simple mapping



If we use this image as a texture

Example (Ex13-1)

- main() in WebGL.js

Load the quad_Z.obj

Load the normal map
The step is the same as loading a 2D texture

```
async function main(){
    canvas = document.getElementById('webgl');
    gl = canvas.getContext('webgl2');
    if(!gl){
        console.log('Failed to get the rendering context for WebGL');
        return ;
    }

    quadObj = await loadOBJtoCreateVBO('quad_Z.obj');

    program = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);
    program.a_Position = gl.getAttribLocation(program, 'a_Position');
    program.a_TexCoord = gl.getAttribLocation(program, 'a_TexCoord');
    program.u_MvpMatrix = gl.getUniformLocation(program, 'u_MvpMatrix');
    program.u_modelMatrix = gl.getUniformLocation(program, 'u_modelMatrix');
    program.u_normalMatrix = gl.getUniformLocation(program, 'u_normalMatrix');
    program.u_LightPosition = gl.getUniformLocation(program, 'u_LightPosition');
    program.u_ViewPosition = gl.getUniformLocation(program, 'u_ViewPosition');
    program.u_Ka = gl.getUniformLocation(program, 'u_Ka');
    program.u_Kd = gl.getUniformLocation(program, 'u_Kd');
    program.u_Ks = gl.getUniformLocation(program, 'u_Ks');
    program.u_Color = gl.getUniformLocation(program, 'u_Color');
    program.u_shininess = gl.getUniformLocation(program, 'u_shininess');
    program.u_Sampler0 = gl.getUniformLocation(program, 'u_Sampler0');

    var normalMapImage = new Image();
    normalMapImage.onload = function(){initTexture(gl, normalMapImage, "normalMapImage");};
    normalMapImage.src = "normalMap.png";

    canvas.onmousedown = function(ev){mouseDown(ev)};
    canvas.onmousemove = function(ev){mouseMove(ev)};
    canvas.onmouseup = function(ev){mouseUp(ev)};
}
```

Example (Ex13-1)

- draw() in WebGL.js

```
function draw(){
  gl.viewport(0, 0, canvas.width, canvas.height);
  gl.clearColor(0.4,0.4,0.4,1);
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);
  gl.enable(gl.DEPTH_TEST);

  let vpMatrix = new Matrix4();
  vpMatrix.setPerspective(70, 1, 1, 100);
  vpMatrix.lookAt(cameraX, cameraY, cameraZ,
                  0, 0, 0,
                  0, 1, 0);

  let mdlMatrix = new Matrix4();
  mdlMatrix.setRotate(angleY, 1, 0, 0);//for mouse rotation
  mdlMatrix.rotate(angleX, 0, 1, 0);//for mouse rotation
  mdlMatrix.scale(4, 4, 4);
  drawOneRegularObject(quadObj, mdlMatrix, vpMatrix, 0.4, 0.4, 0.4);
}
```

Pass the normal map as a texture into the shader

```
function drawOneRegularObject(obj, modelMatrix, vpMatrix, colorR, colorG, colorB){
  gl.useProgram(program);
  let mvpMatrix = new Matrix4();
  let normalMatrix = new Matrix4();
  mvpMatrix.set(vpMatrix);
  mvpMatrix.multiply(modelMatrix);

  //normal matrix
  normalMatrix.setInverseOf(modelMatrix);
  normalMatrix.transpose();

  gl.uniform3f(program.u_LightPosition, lightX, lightY, lightZ);
  gl.uniform3f(program.u_ViewPosition, cameraX, cameraY, cameraZ);
  gl.uniform1f(program.u_Ka, 0.2);
  gl.uniform1f(program.u_Kd, 0.7);
  gl.uniform1f(program.u_Ks, 1.0);
  gl.uniform1f(program.u_shininess, 40.0);
  gl.uniform3f(program.u_Color, colorR, colorG, colorB);
  gl.uniform1i(program.u_Sampler0, 0);
  gl.uniform1i(program.u_Sampler1, 1);

  gl.uniformMatrix4fv(program.u_MvpMatrix, false, mvpMatrix.elements);
  gl.uniformMatrix4fv(program.u_modelMatrix, false, modelMatrix.elements);
  gl.uniformMatrix4fv(program.u_normalMatrix, false, normalMatrix.elements);

  gl.activeTexture(gl.TEXTURE0);
  gl.bindTexture(gl.TEXTURE_2D, textures["normalMapImage"]);

  for( let i=0; i < obj.length; i ++ ){
    initAttributeVariable(gl, program.a_Position, obj[i].vertexBuffer);
    initAttributeVariable(gl, program.a_TexCoord, obj[i].texCoordBuffer);
    gl.drawArrays(gl.TRIANGLES, 0, obj[i].numVertices);
  }
}
```

Example (Ex13-1)

- Shader in WebGL.js

Our normal map

Get the a rgb from the normal map and rescale it from range [0 ~ 1] to [-1 ~ +1]

Transform the normal vector to world space for illumination calculation

Get the normal vector from the normal map and process it

```
var VSHADER_SOURCE = `
    attribute vec4 a_Position;
    attribute vec2 a_TexCoord;
    uniform mat4 u_MvpMatrix;
    uniform mat4 u_modelMatrix;
    uniform mat4 u_normalMatrix;
    varying vec3 v_PositionInWorld;
    varying vec2 v_TexCoord;
    void main(){
        gl_Position = u_MvpMatrix * a_Position;
        v_PositionInWorld = (u_modelMatrix * a_Position).xyz;
        v_TexCoord = a_TexCoord;
    }
`;
```

```
var FSHADER_SOURCE = `
    precision mediump float;
    uniform vec3 u_LightPosition;
    uniform vec3 u_ViewPosition;
    uniform float u_Ka;
    uniform float u_Kd;
    uniform float u_Ks;
    uniform vec3 u_Color;
    uniform float u_shininess;
    uniform sampler2D u_Sampler0;
    uniform highp mat4 u_normalMatrix;
    varying vec3 v_PositionInWorld;
    varying vec2 v_TexCoord;
    void main(){
        // (you can also input them from ouside and make them different)
        vec3 ambientLightColor = u_Color.rgb;
        vec3 diffuseLightColor = u_Color.rgb;
        // assume white specular light (you can also input it from ouside)
        vec3 specularLightColor = vec3(1.0, 1.0, 1.0);

        vec3 ambient = ambientLightColor * u_Ka;

        //normal vector from normal map
        vec3 nMapNormal = texture2D( u_Sampler0, v_TexCoord ).rgb * 2.0 - 1.0;
        vec3 n = normalize( nMapNormal );
        vec3 normal = normalize( vec3( u_normalMatrix * vec4( n, 1.0) ) );

        vec3 lightDirection = normalize(u_LightPosition - v_PositionInWorld);
        float nDotL = max(dot(lightDirection, normal), 0.0);
        vec3 diffuse = diffuseLightColor * u_Kd * nDotL;

        vec3 specular = vec3(0.0, 0.0, 0.0);
        if(nDotL > 0.0) {
            vec3 R = reflect(-lightDirection, normal);

            vec3 V = normalize(u_ViewPosition - v_PositionInWorld);
            float specAngle = clamp(dot(R, V), 0.0, 1.0);
            specular = u_Ks * pow(specAngle, u_shininess) * specularLightColor;
        }

        gl_FragColor = vec4( ambient + diffuse + specular, 1.0 );
    }
`;
```

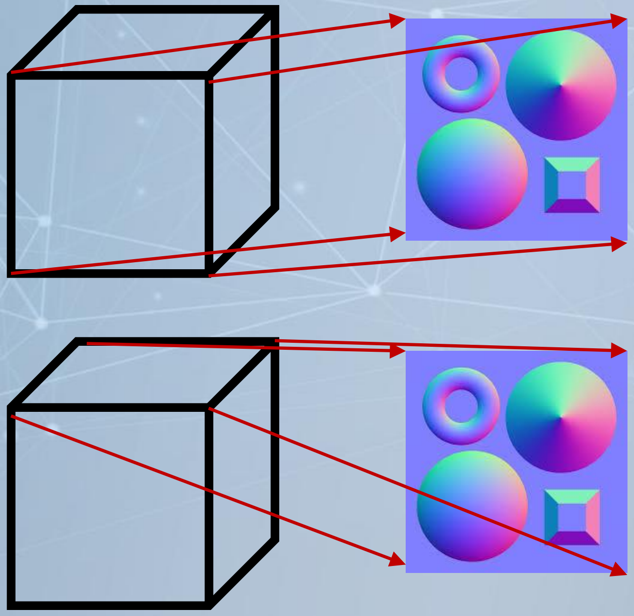
Try and Think (5mins)

- Download the code and run it

- Can we directly use this code and apply the normal map to each face of a **cube** object?
  - I put a cube.obj in the folder
  - You can modify “quadObj = await loadOBJtoCreateVBO('**quad_Z.obj**’);” to “quadObj = await loadOBJtoCreateVBO(“**cube.obj**’);”
  - And, modify “mdlMatrix.scale(4, 4, 4);” to “mdlMatrix.scale(2, 2, 2);”
  - Run it and rotate it

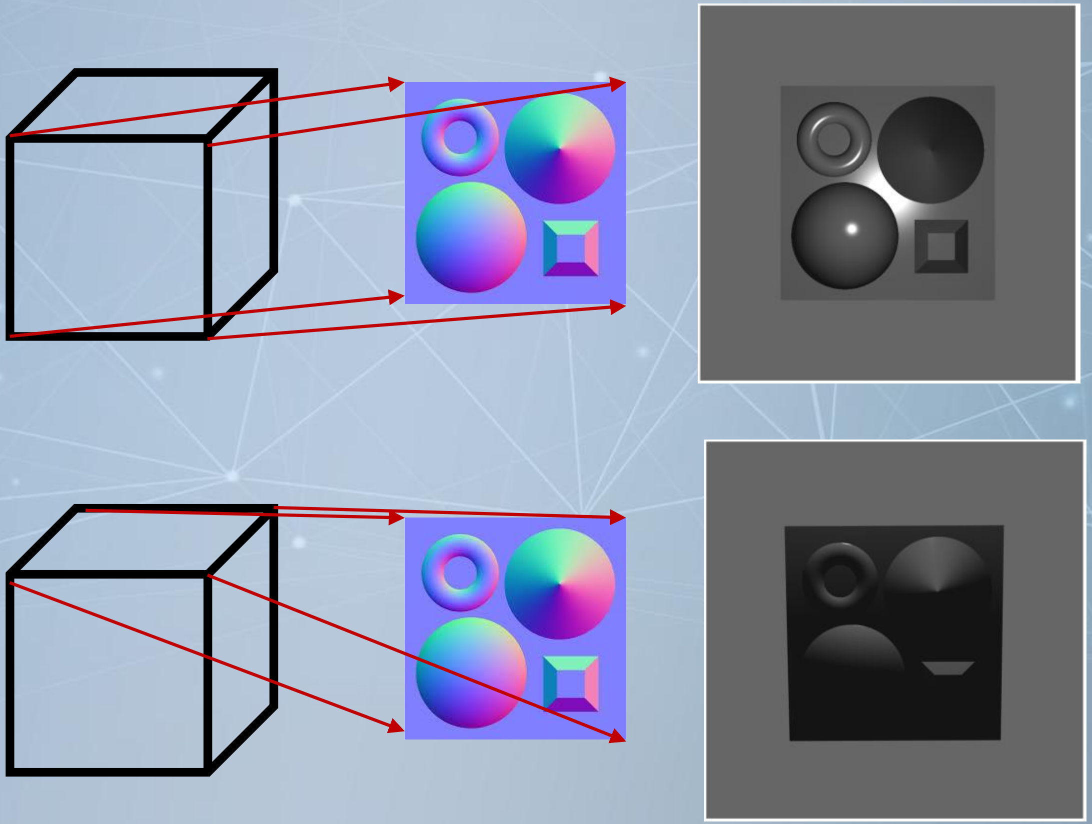
The Normal Map on a Cube

- Each face has the same texture coordinate
- Use the same normal map on each face



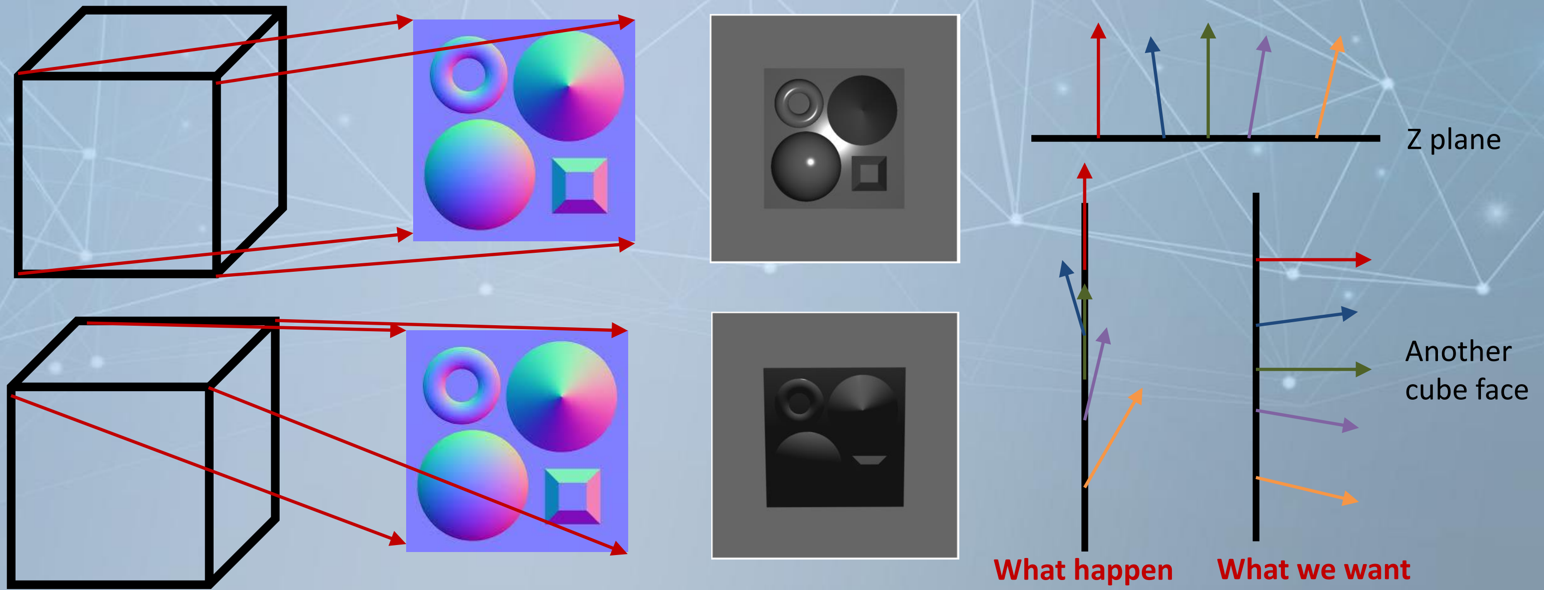
```
# cube.obj
#
mtllib cube.mtl
o cube
v -1.000000 -1.000000 1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 1.000000 1.000000
v 1.000000 1.000000 1.000000
v -1.000000 1.000000 -1.000000
v 1.000000 1.000000 -1.000000
v -1.000000 -1.000000 -1.000000
v 1.000000 -1.000000 -1.000000
vt 0.000000 0.000000
vt 1.000000 0.000000
vt 0.000000 1.000000
vt 1.000000 1.000000
vn 0.000000 0.000000 1.000000
vn 0.000000 1.000000 0.000000
vn 0.000000 0.000000 -1.000000
vn 0.000000 -1.000000 0.000000
vn 1.000000 0.000000 0.000000
vn -1.000000 0.000000 0.000000
usemtl cube
s 1
f 1/1/1 2/2/1 3/3/1
f 3/3/1 2/2/1 4/4/1
f 3/1/2 4/2/2 5/3/2
f 5/3/2 4/2/2 6/4/2
f 5/4/3 6/3/3 7/2/3
f 7/2/3 6/3/3 8/1/3
f 7/1/4 8/2/4 1/3/4
f 1/3/4 8/2/4 2/4/4
f 2/1/5 8/2/5 4/3/5
f 4/3/5 8/2/5 6/4/5
f 7/1/6 1/2/6 5/3/6
f 5/3/6 1/2/6 3/4/6
```

The Normal Map on a Cube



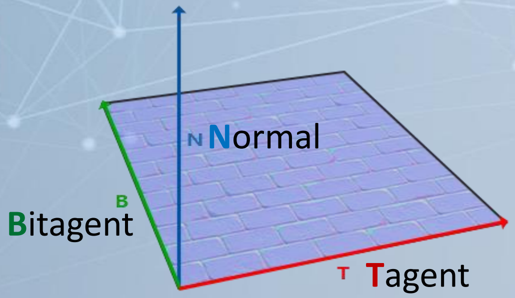
The Normal Map on a Cube

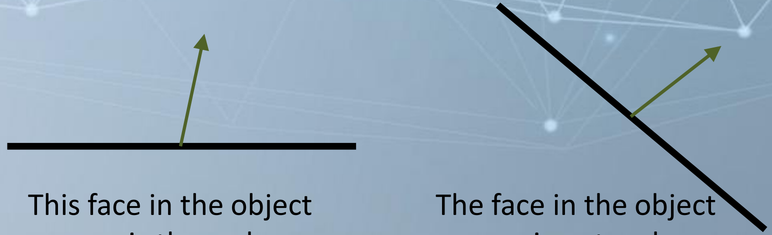
- Why?
  - Remember: the projection plane (for normal map) is **z plane**, so most of the normal vectors are close to [0,0,1]



Normal Vector from Tangent Space to Object Space

- The normal vectors in normal map are defined in a space called **"tangent space"**
  - Tangent space is the space local to the triangle surface
  - Or, in this normal map application, you can imagine it is the space of the normal map
- Other explanation
  - The vector in normal map is only correct for the z-plane (and face to z+ direction).
  - If your triangle face is in not this case, you have to **rotate** the normal vector by your self
  - This is the step to transform a normal vector in the normal map **from tangent space to object space**

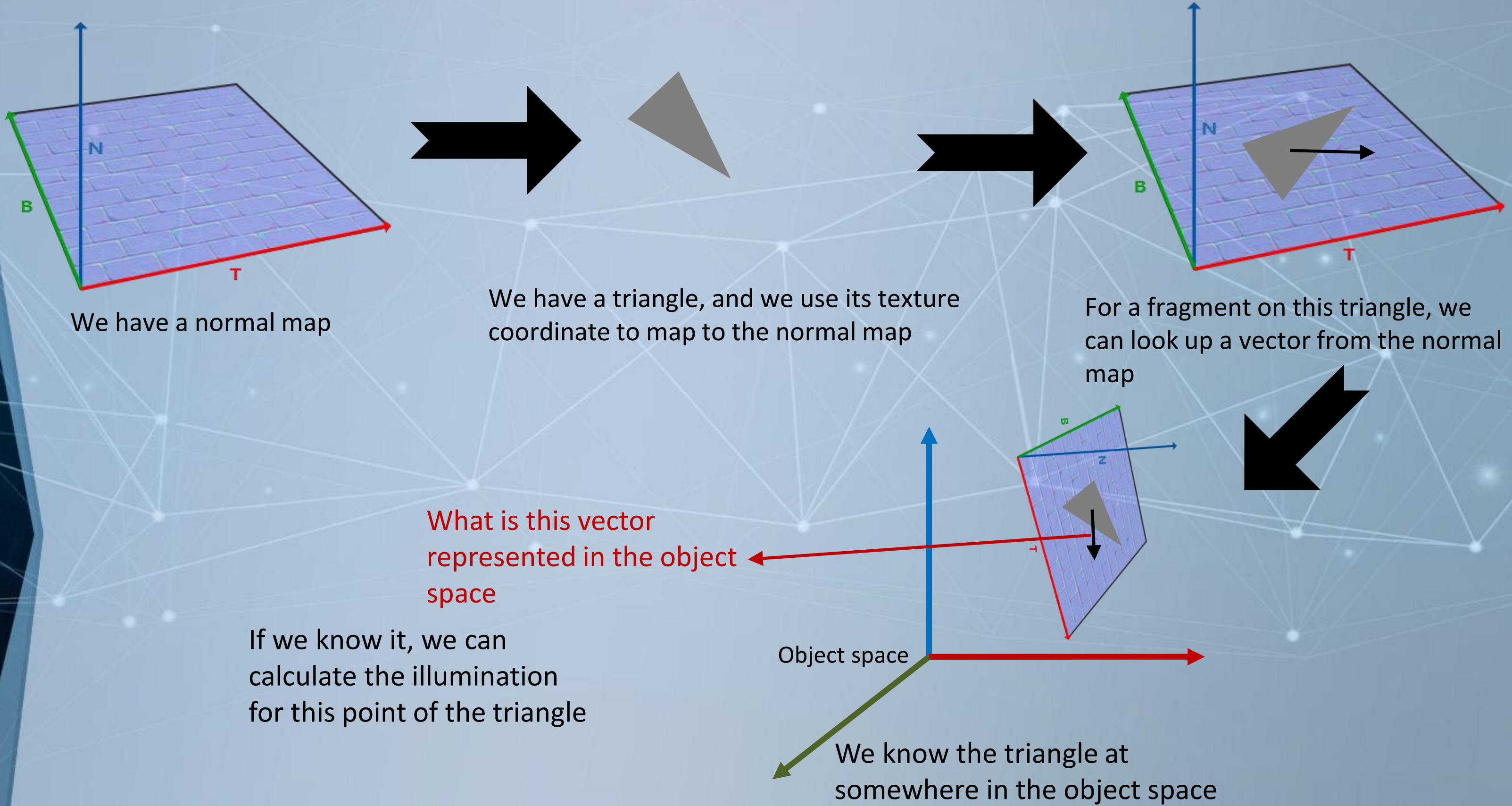




This face in the object space is the z-plane

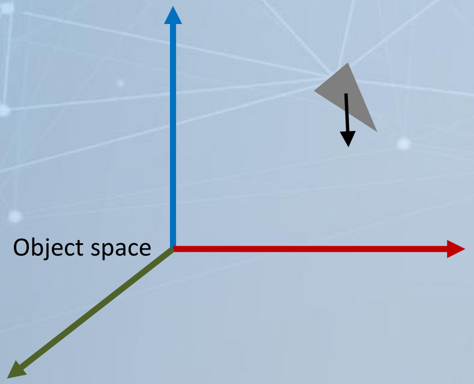
The face in the object space is not z-plane

Normal Vector from Tangent Space to Object Space



Normal Vector from Tangent Space to Object Space

- How to transform a vector from the normal map to the object space?
  - A vector, $[x_t, y_t, z_t]$, from the normal map is defined in the tangent space
  - We also know where the triangle is in the object space
  - This question is equivalent to asking that what the $[x_o, y_o, z_o]$ is ($[x_o, y_o, z_o]$ is the coordinate of the vector $[x_t, y_t, z_t]$ in the object space

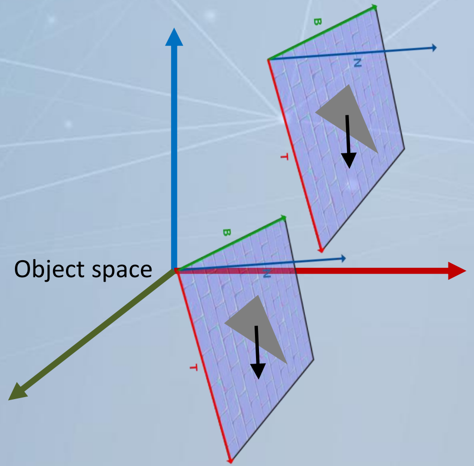


- How?
- Use "TBN" matrix
- TBN is a matrix to transform a vector/position from tangent space to object space
  - $$\begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} = TBN * \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix}$$

If the vector is represented in "object space"
, we can use it to calculate illumination

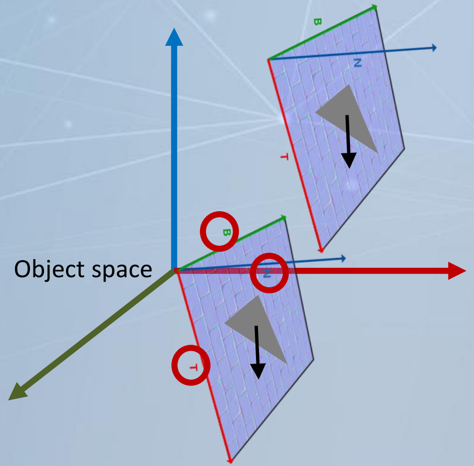
TBN Matrix Calculation

- Note: only the rotation transformation (ignore translation) between object space and tangent space matters
  - If two triangle surfaces parallel with each other, their normal vectors are the same



TBN Matrix Calculation

- Note: only the rotation transformation (ignore translation) between object space and tangent space matters
  - If two triangle surfaces parallel with each other, their normal vectors are the same



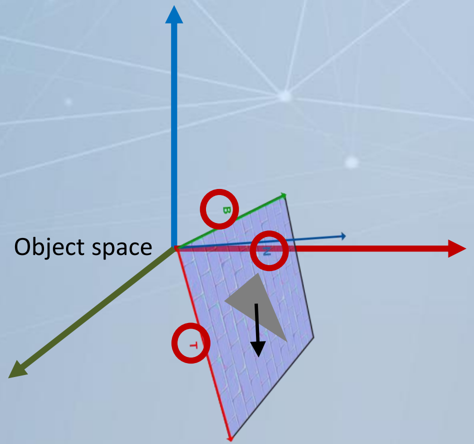
If we know the three basis unit vectors "T", "B", and "N" represented in the object space, we can have the change-of-basis matrix, TBN matrix.

$$TBN = \begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix}$$

$$\begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} = TBN * \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix}$$

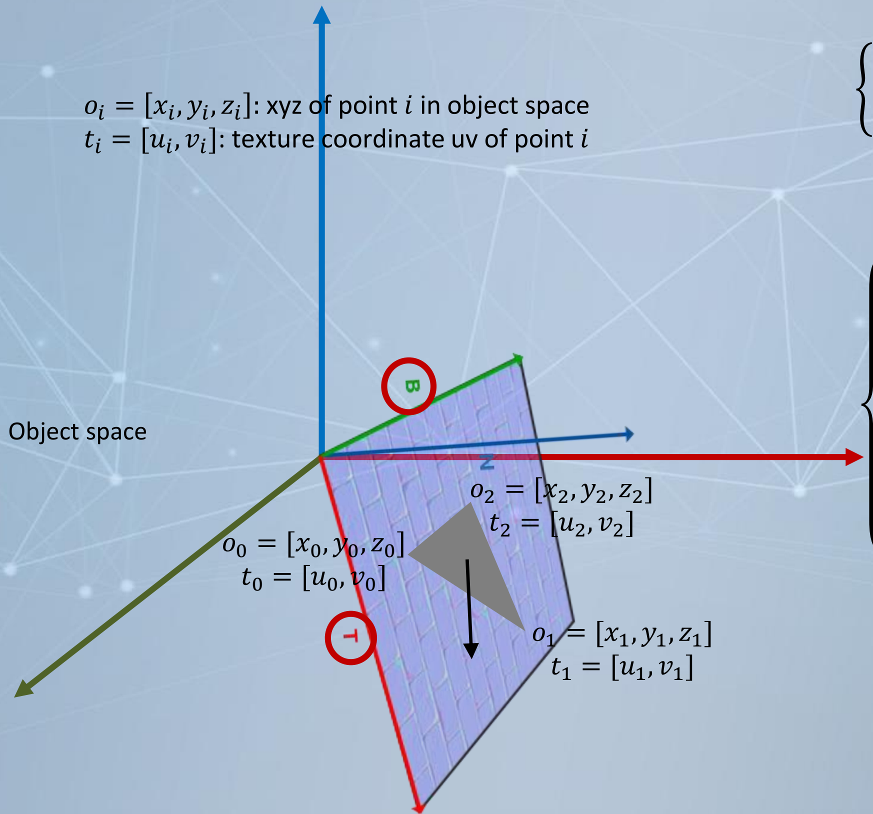
TBN Matrix Calculation

- What we know?
  - The xyz of the three vertices of the triangle in object space and their texture coordinates (how to map them to the tangent space or normal map space)
- What we want to calculate?
  - The T and B unit vectors in the object space
  - N? $T \times B$ (cross product)



TBN Matrix Calculation

$o_i = [x_i, y_i, z_i]$: xyz of point i in object space
$t_i = [u_i, v_i]$: texture coordinate uv of point i



$$\begin{cases} o_1 - o_0 = (u_1 - u_0) * T + (v_1 - v_0) * B \\ o_2 - o_0 = (u_2 - u_0) * T + (v_2 - v_0) * B \end{cases}$$

$$\Downarrow$$

$$\begin{cases} x_1 - x_0 = (u_1 - u_0) * T_x + (v_1 - v_0) * B_x \\ y_1 - y_0 = (u_1 - u_0) * T_y + (v_1 - v_0) * B_y \\ z_1 - z_0 = (u_1 - u_0) * T_z + (v_1 - v_0) * B_z \\ x_2 - x_0 = (u_2 - u_0) * T_x + (v_2 - v_0) * B_x \\ y_2 - y_0 = (u_2 - u_0) * T_y + (v_2 - v_0) * B_y \\ z_2 - z_0 = (u_2 - u_0) * T_z + (v_2 - v_0) * B_z \end{cases}$$

We want to calculate T and B (6 unknowns).
We have six equations, so we can solve it.

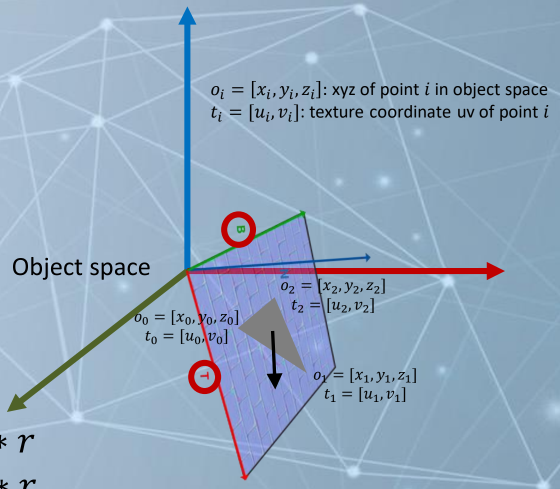
Summary of TBN Calculation (1/3)

- After loading the normal image and before using the data (rgb) as a vector, remember rescale data range from [0 ~ 1] to [-1 ~ +1]

$$\begin{bmatrix} r \\ g \\ b \end{bmatrix} * 2.0 - 1.0 = \text{a vector}$$

Summary of TBN Calculation (2/3)

- You need to calculate TBN matrix to transform a vector from normal to object space for illumination
  - Solve the six equations to get T and B
- Here is the algorithm to calculate T and B
  - $\text{delta}O_1 = o_1 - o_0$
  - $\text{delta}O_2 = o_2 - o_0$
  - $\text{delta}T_1 = t_1 - t_0$
  - $\text{delta}T_2 = t_2 - t_0$
  - $r = \dfrac{1.0}{\text{delta}T_1.x * \text{delta}T_2.y - \text{delta}T_1.y * \text{delta}T_2.x}$
  - $T = (\text{delta}O_1 * \text{delta}T_2.y - \text{delta}O_2 * \text{delta}T_1.y) * r$
  - $B = (\text{delta}O_2 * \text{delta}T_1.x - \text{delta}O_1 * \text{delta}T_2.x) * r$
  - $N = T \times B$
  - T, B and N should be normalized to unit vector before make the TBN matrix
  - $TBN = \begin{bmatrix} T_x & B_x & N_x \\ T_y & B_y & N_y \\ T_z & B_z & N_z \end{bmatrix}$



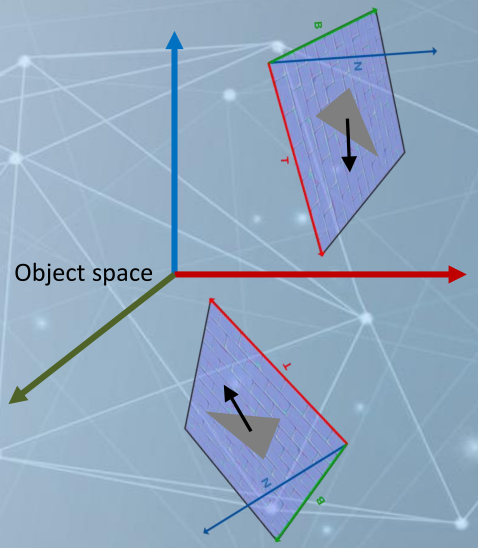
$$\begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} = TBN * \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix}$$

Object space xyz

Tangent space xyz

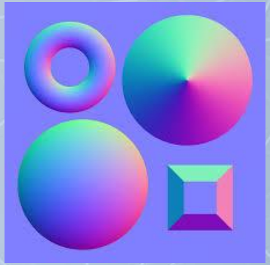
Summary of TBN Calculation (3/3)

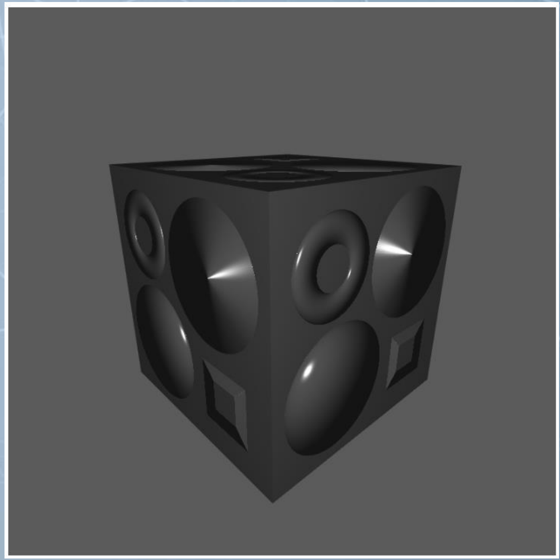
- Each triangle face has its own TBN matrix to transform a vector to object space
- In the implementation, we can calculate T and B on each vertex
  - Of course, the vertices of same triangle will have the same T and B
- Pass T and B to attribute variables
- Calculate N and TBN matrix in shader and use TBN before calculating illumination



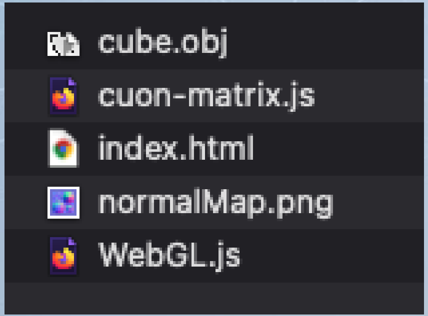
Example (Ex13-2)

- Apply this normal map on each face of a cube object





- Files:



Example (Ex13-2)

- main() in WebGL.js

We will calculate T and B vectors for each vertices and pass them into shader

```javascript
async function main(){
    canvas = document.getElementById('webgl');
    gl = canvas.getContext('webgl2');
    if(!gl){
        console.log('Failed to get the rendering context for WebGL');
        return ;
    }

    cubeObj = await loadOBJtoCreateVBO('cube.obj');

    program = compileShader(gl, VSHADER_SOURCE, FSHADER_SOURCE);
    program.a_Position = gl.getAttribLocation(program, 'a_Position');
    program.a_TexCoord = gl.getAttribLocation(program, 'a_TexCoord');
    program.a_Tagent = gl.getAttribLocation(program, 'a_Tagent');
    program.a_Bitagent = gl.getAttribLocation(program, 'a_Bitagent');
    program.u_MvpMatrix = gl.getUniformLocation(program, 'u_MvpMatrix');
    program.u_modelMatrix = gl.getUniformLocation(program, 'u_modelMatrix');
    program.u_normalMatrix = gl.getUniformLocation(program, 'u_normalMatrix');
    program.u_LightPosition = gl.getUniformLocation(program, 'u_LightPosition');
    program.u_ViewPosition = gl.getUniformLocation(program, 'u_ViewPosition');
    program.u_Ka = gl.getUniformLocation(program, 'u_Ka');
    program.u_Kd = gl.getUniformLocation(program, 'u_Kd');
    program.u_Ks = gl.getUniformLocation(program, 'u_Ks');
    program.u_Color = gl.getUniformLocation(program, 'u_Color');
    program.u_shininess = gl.getUniformLocation(program, 'u_shininess');
    program.u_Sampler0 = gl.getUniformLocation(program, 'u_Sampler0');

    var normalMapImage = new Image();
    normalMapImage.onload = function(){initTexture(gl, normalMapImage, "normalMapImage");};
    normalMapImage.src = "normalMap.png";

    canvas.onmousedown = function(ev){mouseDown(ev)};
    canvas.onmousemove = function(ev){mouseMove(ev)};
    canvas.onmouseup = function(ev){mouseUp(ev)};
}
```

Example (Ex13-2)

- calculateTangentSpace() in WebGL.js
  - This is the algorithm we mention in "Summary of TBN Calculation (2/3)"
  - When we load the vertices and texture coordinates of a 3D model, we pass them into this function and this function calculates T and B vectors for each vertex
  - Return T and B vectors of all vertices

```
async function loadOBJtoCreateVBO( objFile ){
  let objComponents = [];
  response = await fetch(objFile);
  text = await response.text();
  obj = parseOBJ(text);
  for( let i=0; i < obj.geometries.length; i ++ ){
    let tagentSpace = calculateTangentSpace(obj.geometries[i].data.position,
                                            obj.geometries[i].data.texcoord);
    let o = initVertexBufferForLaterUse(gl,
                                        obj.geometries[i].data.position,
                                        obj.geometries[i].data.normal,
                                        obj.geometries[i].data.texcoord,
                                        tagentSpace.tagents,
                                        tagentSpace.bitagents);
    objComponents.push(o);
  }
  return objComponents;
}
```

```
function calculateTangentSpace(position, texcoord){
  //iterate through all triangles
  let tagents = [];
  let bitagents = [];
  for( let i = 0; i < position.length/9; i++ ){
    let v00 = position[i*9 + 0];
    let v01 = position[i*9 + 1];
    let v02 = position[i*9 + 2];
    let v10 = position[i*9 + 3];
    let v11 = position[i*9 + 4];
    let v12 = position[i*9 + 5];
    let v20 = position[i*9 + 6];
    let v21 = position[i*9 + 7];
    let v22 = position[i*9 + 8];
    let uv00 = texcoord[i*6 + 0];
    let uv01 = texcoord[i*6 + 1];
    let uv10 = texcoord[i*6 + 2];
    let uv11 = texcoord[i*6 + 3];
    let uv20 = texcoord[i*6 + 4];
    let uv21 = texcoord[i*6 + 5];

    let deltaPos10 = v10 - v00;
    let deltaPos11 = v11 - v01;
    let deltaPos12 = v12 - v02;
    let deltaPos20 = v20 - v00;
    let deltaPos21 = v21 - v01;
    let deltaPos22 = v22 - v02;

    let deltaUV10 = uv10 - uv00;
    let deltaUV11 = uv11 - uv01;
    let deltaUV20 = uv20 - uv00;
    let deltaUV21 = uv21 - uv01;

    let r = 1.0 / (deltaUV10 * deltaUV21 - deltaUV11 * deltaUV20);
    let tangentX = (deltaPos10 * deltaUV21   - deltaPos20 * deltaUV11)*r;
    let tangentY = (deltaPos11 * deltaUV21   - deltaPos21 * deltaUV11)*r;
    let tangentZ = (deltaPos12 * deltaUV21   - deltaPos22 * deltaUV11)*r;
    for( let j = 0; j < 3; j++ ){
      tagents.push(tangentX);
      tagents.push(tangentY);
      tagents.push(tangentZ);
    }
    let bitangentX = (deltaPos20 * deltaUV10   - deltaPos10 * deltaUV20)*r;
    let bitangentY = (deltaPos21 * deltaUV10   - deltaPos11 * deltaUV20)*r;
    let bitangentZ = (deltaPos22 * deltaUV10   - deltaPos12 * deltaUV20)*r;
    for( let j = 0; j < 3; j++ ){
      bitagents.push(bitangentX);
      bitagents.push(bitangentY);
      bitagents.push(bitangentZ);
    }
  }
  let obj = {};
  obj['tagents'] = tagents;
  obj['bitagents'] = bitagents;
  return obj;
}
```

Example (Ex13-2)

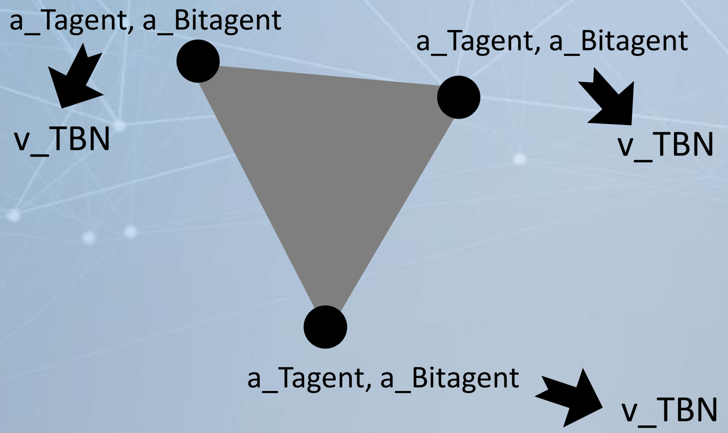
- drawOneRegularObject() in WebGL.js

To draw the cube, we pass vertices, texture coordinates, T and B vectors to shaders

```
function drawOneRegularObject(obj, modelMatrix, vpMatrix, colorR, colorG, colorB){
  gl.useProgram(program);
  let mvpMatrix = new Matrix4();
  let normalMatrix = new Matrix4();
  mvpMatrix.set(vpMatrix);
  mvpMatrix.multiply(modelMatrix);

  //normal matrix
  normalMatrix.setInverseOf(modelMatrix);
  normalMatrix.transpose();

  gl.uniform3f(program.u_LightPosition, lightX, lightY, lightZ);
  gl.uniform3f(program.u_ViewPosition, cameraX, cameraY, cameraZ);
  gl.uniform1f(program.u_Ka, 0.2);
  gl.uniform1f(program.u_Kd, 0.7);
  gl.uniform1f(program.u_Ks, 1.0);
  gl.uniform1f(program.u_shininess, 40.0);
  gl.uniform3f(program.u_Color, colorR, colorG, colorB);
  gl.uniform1i(program.u_Sampler0, 0);
  gl.uniform1i(program.u_Sampler1, 1);

  gl.uniformMatrix4fv(program.u_MvpMatrix, false, mvpMatrix.elements);
  gl.uniformMatrix4fv(program.u_modelMatrix, false, modelMatrix.elements);
  gl.uniformMatrix4fv(program.u_normalMatrix, false, normalMatrix.elements);

  gl.activeTexture(gl.TEXTURE0);
  gl.bindTexture(gl.TEXTURE_2D, textures["normalMapImage"]);

  for( let i=0; i < obj.length; i ++ ){
    initAttributeVariable(gl, program.a_Position, obj[i].vertexBuffer);
    initAttributeVariable(gl, program.a_TexCoord, obj[i].texCoordBuffer);
    initAttributeVariable(gl, program.a_Tagent, obj[i].tagentsBuffer);
    initAttributeVariable(gl, program.a_Bitagent, obj[i].bitagentsBuffer);
    gl.drawArrays(gl.TRIANGLES, 0, obj[i].numVertices);
  }
}
```

Example (Ex13-2)

- Vertex shader in WebGL.js
- Although each vertex of this triangle has its own T and B vectors, they are all the same. So TBN matrix of these three vertices are the same.
- So, v_TBN of any fragment in this triangle in the fragment shader is also the same




```
var VSHADER_SOURCE = `
    attribute vec4 a_Position;
    attribute vec2 a_TexCoord;
    attribute vec3 a_Tagent;
    attribute vec3 a_Bitagent;
    uniform mat4 u_MvpMatrix;
    uniform mat4 u_modelMatrix;
    uniform mat4 u_normalMatrix;
    varying vec3 v_PositionInWorld;
    varying vec2 v_TexCoord;
    varying mat4 v_TBN;
    void main(){
        gl_Position = u_MvpMatrix * a_Position;
        v_PositionInWorld = (u_modelMatrix * a_Position).xyz;
        v_TexCoord = a_TexCoord;
        //create TBN matrix
        vec3 tagent = normalize(a_Tagent);
        vec3 bitagent = normalize(a_Bitagent);
        vec3 nVector = cross(tagent, bitagent);
        v_TBN = mat4(tagent.x, tagent.y, tagent.z, 0.0,
                     bitagent.x, bitagent.y, bitagent.z, 0.0,
                     nVector.x, nVector.y, nVector.z, 0.0,
                     0.0, 0.0, 0.0, 1.0);
    }
`;
```

Example (Ex13-2)

- Vertex shader in WebGL.js

$$\begin{bmatrix} tagent.x & bitagent.x & nVector.x & 0.0 \\ tagent.y & bitagent.y & nVector.y & 0.0 \\ tagent.z & bitagent.z & nVector.z & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

```
var VSHADER_SOURCE = `
    attribute vec4 a_Position;
    attribute vec2 a_TexCoord;
    attribute vec3 a_Tagent;
    attribute vec3 a_Bitagent;
    uniform mat4 u_MvpMatrix;
    uniform mat4 u_modelMatrix;
    uniform mat4 u_normalMatrix;
    varying vec3 v_PositionInWorld;
    varying vec2 v_TexCoord;
    varying mat4 v_TBN;
    void main(){
        gl_Position = u_MvpMatrix * a_Position;
        v_PositionInWorld = (u_modelMatrix * a_Position).xyz;
        v_TexCoord = a_TexCoord;
        //create TBN matrix
        vec3 tagent = normalize(a_Tagent);
        vec3 bitagent = normalize(a_Bitagent);
        vec3 nVector = cross(tagent, bitagent);
        v_TBN = mat4(tagent.x, tagent.y, tagent.z, 0.0,
                     bitagent.x, bitagent.y, bitagent.z, 0.0,
                     nVector.x, nVector.y, nVector.z, 0.0,
                     0.0, 0.0, 0.0, 1.0);
    }
`;
```

Example (Ex13-2)

- Fragment shader in WebGL.js

get and calculate a vector from the normal map

```
var FSHADER_SOURCE = `
    precision mediump float;
    uniform vec3 u_LightPosition;
    uniform vec3 u_ViewPosition;
    uniform float u_Ka;
    uniform float u_Kd;
    uniform float u_Ks;
    uniform vec3 u_Color;
    uniform float u_shininess;
    uniform sampler2D u_Sampler0;
    uniform highp mat4 u_normalMatrix;
    varying vec3 v_PositionInWorld;
    varying vec2 v_TexCoord;
    varying mat4 v_TBN;
    void main(){
        // (you can also input them from ouside and make them different)
        vec3 ambientLightColor = u_Color.rgb;
        vec3 diffuseLightColor = u_Color.rgb;
        // assume white specular light (you can also input it from ouside)
        vec3 specularLightColor = vec3(1.0, 1.0, 1.0);

        vec3 ambient = ambientLightColor * u_Ka;

        //normal vector from normal map
        vec3 nMapNormal = normalize( texture2D( u_Sampler0, v_TexCoord ).rgb * 2.0 - 1.0 );
        vec3 normal = normalize( vec3( u_normalMatrix * v_TBN * vec4( nMapNormal, 1.0) ) );

        vec3 lightDirection = normalize(u_LightPosition - v_PositionInWorld);
        float nDotL = max(dot(lightDirection, normal), 0.0);
        vec3 diffuse = diffuseLightColor * u_Kd * nDotL;

        vec3 specular = vec3(0.0, 0.0, 0.0);
        if(nDotL > 0.0) {
            vec3 R = reflect(-lightDirection, normal);

            vec3 V = normalize(u_ViewPosition - v_PositionInWorld);
            float specAngle = clamp(dot(R, V), 0.0, 1.0);
            specular = u_Ks * pow(specAngle, u_shininess) * specularLightColor;
        }

        gl_FragColor = vec4( ambient + diffuse + specular, 1.0 );
    }
`;
```

Transform vector to world space

Transform the vector from tangent space to object space

Try and Think (5mins)

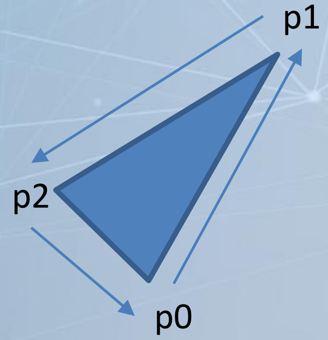
- Download and run it

- Can this version of code work for all 3D models?
  - for example, sonic.obj?
  - I have put sonic.obj in the folder.
    - You can modify “cubeObj = await loadOBJtoCreateVBO('cube.obj’);” to “cubeObj = await loadOBJtoCreateVBO(‘sonic.obj’);”
    - And modify “mdlMatrix.scale(2, 2, 2);” to “mdlMatrix.scale(0.18, 0.18, 0.18);”

What Happen on Sonic.obj?

- When you look at a triangle of a 3D object from outside, the vertices of the triangle is stored (and pass to shader) by “counter-clockwise order”.
  - This “counter-clockwise order” is useful for WebGL to check the face is a front or back face. But we do not emphasize this scheme.

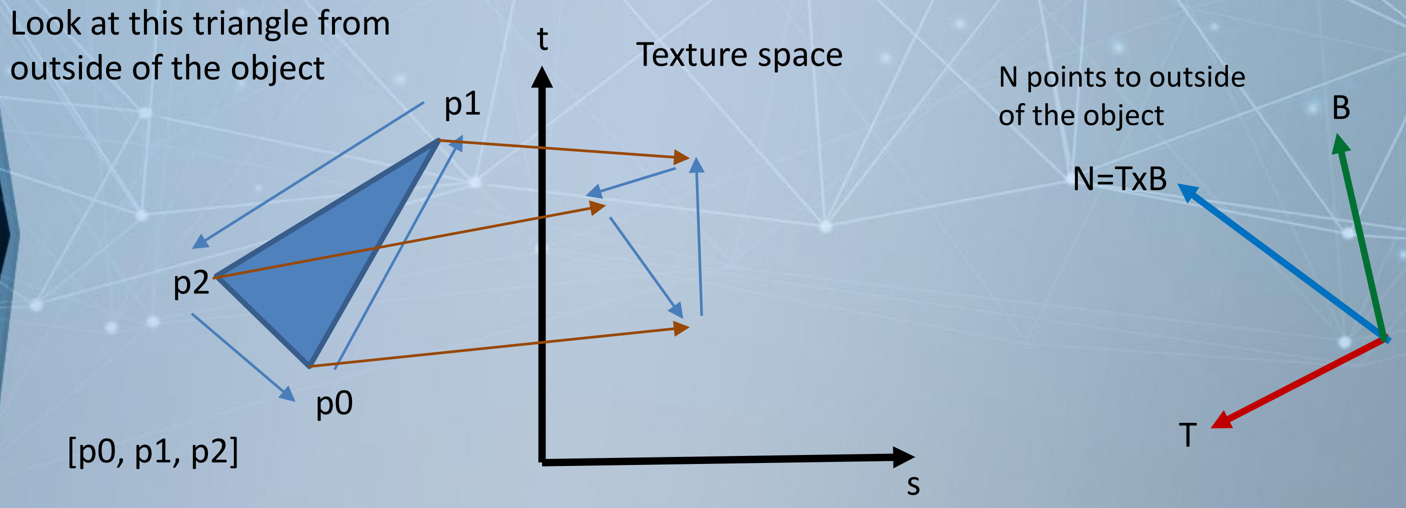
Look at this triangle from outside of the object



[p0, p1, p2]

What Happen on Sonic.obj?

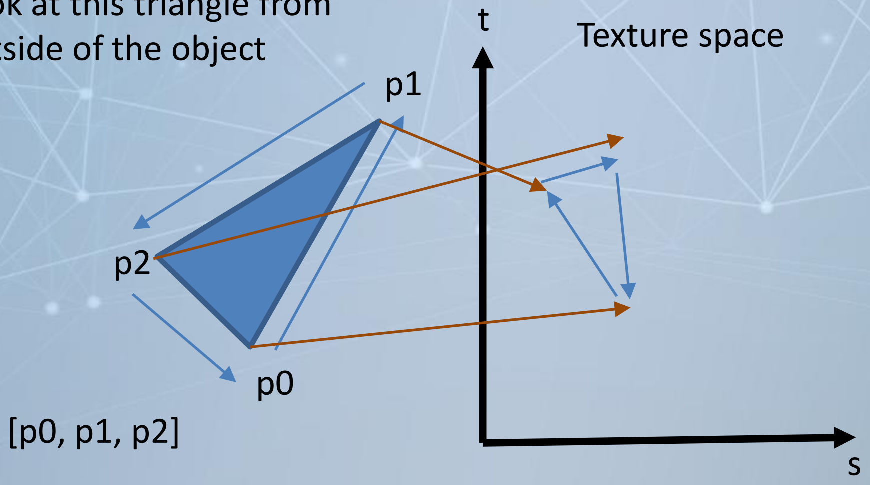
- If the order of texture coordinates of P0, P1, P2 in the texture space is also counter-clockwise, we just use $\boldsymbol{T} \times \boldsymbol{B}$ **to calculate N vector**. This is **no problem**.
  - N will point to outside of the object and vector from the normal map will point to outside of the object as well.



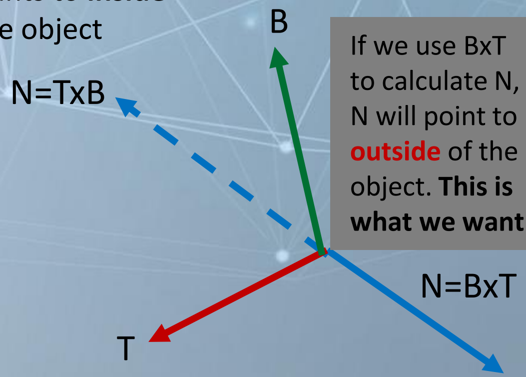
What Happen on Sonic.obj?

- If the order of texture coordinates of P0, P1, P2 in the texture space is **clockwise**, $T \times B$ will point to **inside** of the object and the vector from normal map will point to **inside** of the object as well.
  - We use the vector as normal vector to calculate illumination, so we won't get the the result we expect

Look at this triangle from outside of the object



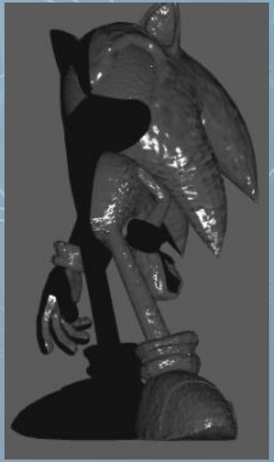
N points to **inside** of the object



If we use BxT to calculate N, N will point to **outside** of the object. **This is what we want**

What Happen on Sonic.obj?

- sonic.obj has 13 components
- The orders of texture coordinates of the 3 vertices of a triangle of some components in the texture space are counter-clockwise. But that of the other components are clockwise.

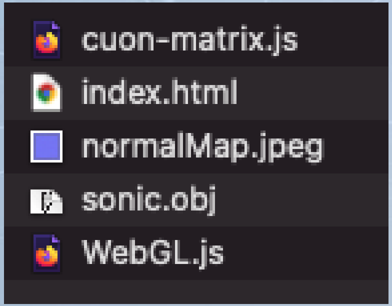


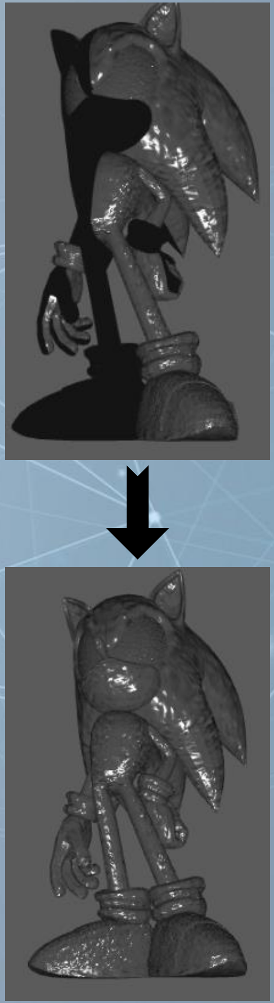
- What we can do?
  - Check the order of texture coordinates of each triangle
  - If it is counter-clockwise in texture space, use $T \times B$ to calculate N
  - If it is clockwise in texture space, use $B \times T$ to calculate N

Example (Ex13-3)

- Deal with the arbitrary texture coordinate order problem

- Files:





Example (Ex13-3)

- calculateTangentSpace() in WebGL.js
- If we have a vertex p_0, and p_1 and p_2 are the other two vertices on the same triangle.
- They are stored by this order $[p_0, p_1, p_2]$.
- Their corresponding texture coordinates are $[uv_0, uv_1, uv_2]$.
  - We will calculate cross produce, $(uv_1 - uv_0) \times (uv_2 - uv_0)$
  - If the cross produce is positive, $[uv_0, uv_1, uv_2]$ is in counter-clockwise order. We should calculate N by $T \times B$
  - If the cross produce is positive, $[uv_0, uv_1, uv_2]$ is in clockwise order. We should calculate N by $B \times T$

```
function calculateTangentSpace(position, texcoord){
  //iterate through all triangles
  let tagents = [];
  let bitagents = [];
  let crossTexCoords = [];
  for( let i = 0; i < position.length/9; i++ ){
    let v00 = position[i*9 + 0];
    let v01 = position[i*9 + 1];
    let v02 =  (parameter) position: any
    let v10 = position[i*9 + 3];
    let v11 = position[i*9 + 4];
    let v12 = position[i*9 + 5];
    let v20 = position[i*9 + 6];
    let v21 = position[i*9 + 7];
    let v22 = position[i*9 + 8];
    let uv00 = texcoord[i*6 + 0];
    let uv01 = texcoord[i*6 + 1];
    let uv10 = texcoord[i*6 + 2];
    let uv11 = texcoord[i*6 + 3];
    let uv20 = texcoord[i*6 + 4];
    let uv21 = texcoord[i*6 + 5];

    let deltaPos10 = v10 - v00;
    let deltaPos11 = v11 - v01;
    let deltaPos12 = v12 - v02;
    let deltaPos20 = v20 - v00;
    let deltaPos21 = v21 - v01;
    let deltaPos22 = v22 - v02;

    let deltaUV10 = uv10 - uv00;
    let deltaUV11 = uv11 - uv01;
    let deltaUV20 = uv20 - uv00;
    let deltaUV21 = uv21 - uv01;

    let r = 1.0 / (deltaUV10 * deltaUV21 - deltaUV11 * deltaUV20);
    for( let j=0; j< 3; j++ ){
      crossTexCoords.push( (deltaUV10 * deltaUV21 - deltaUV11 * deltaUV20) );
    }
    let tangentX = (deltaPos10 * deltaUV21 - deltaPos20 * deltaUV11)*r;
    let tangentY = (deltaPos11 * deltaUV21 - deltaPos21 * deltaUV11)*r;
    let tangentZ = (deltaPos12 * deltaUV21 - deltaPos22 * deltaUV11)*r;
    for( let j = 0; j < 3; j++ ){
      tagents.push(tangentX);
      tagents.push(tangentY);
      tagents.push(tangentZ);
    }
    let bitangentX = (deltaPos20 * deltaUV10 - deltaPos10 * deltaUV20)*r;
    let bitangentY = (deltaPos21 * deltaUV10 - deltaPos11 * deltaUV20)*r;
    let bitangentZ = (deltaPos22 * deltaUV10 - deltaPos12 * deltaUV20)*r;
    for( let j = 0; j < 3; j++ ){
      bitagents.push(bitangentX);
      bitagents.push(bitangentY);
      bitagents.push(bitangentZ);
    }
  }
  let obj = {};
  obj['tagents'] = tagents;
  obj['bitagents'] = bitagents;
  obj['crossTexCoords'] = crossTexCoords;
  return obj;
```

Example (Ex13-3)

- calculateTangentSpace() in WebGL.js

- If we have a vertex p_0, and p_1 and p_2 are the other two vertices on the same triangle.
- They are stored by this order $[p_0, p_1, p_2]$.
- Their corresponding texture coordinates are $[uv_0, uv_1, uv_2]$.
  - We will calculate cross produce, $(uv_1 - uv_0) \times (uv_2 - uv_0)$
  - If the cross produce is positive, $[uv_0, uv_1, uv_2]$ is in counter-clockwise order. We should calculate N by $T \times B$
  - If the cross produce is positive, $[uv_0, uv_1, uv_2]$ is in clockwise order. We should calculate N by $B \times T$

We will return the cross products of all vertices and pass them into an attribute variable in shader. We will check and do this in vertex shader.

```
function calculateTangentSpace(position, texcoord){
  //iterate through all triangles
  let tagents = [];
  let bitagents = [];
  let crossTexCoords = [];
  for( let i = 0; i < position.length/9; i++ ){
    let v00 = position[i*9 + 0];
    let v01 = position[i*9 + 1];
    let v02 = (parameter) position: any
    let v10 = position[i*9 + 3];
    let v11 = position[i*9 + 4];
    let v12 = position[i*9 + 5];
    let v20 = position[i*9 + 6];
    let v21 = position[i*9 + 7];
    let v22 = position[i*9 + 8];
    let uv00 = texcoord[i*6 + 0];
    let uv01 = texcoord[i*6 + 1];
    let uv10 = texcoord[i*6 + 2];
    let uv11 = texcoord[i*6 + 3];
    let uv20 = texcoord[i*6 + 4];
    let uv21 = texcoord[i*6 + 5];

    let deltaPos10 = v10 - v00;
    let deltaPos11 = v11 - v01;
    let deltaPos12 = v12 - v02;
    let deltaPos20 = v20 - v00;
    let deltaPos21 = v21 - v01;
    let deltaPos22 = v22 - v02;

    let deltaUV10 = uv10 - uv00;
    let deltaUV11 = uv11 - uv01;
    let deltaUV20 = uv20 - uv00;
    let deltaUV21 = uv21 - uv01;

    let r = 1.0 / (deltaUV10 * deltaUV21 - deltaUV11 * deltaUV20);
    for( let j=0; j< 3; j++ ){
      crossTexCoords.push( (deltaUV10 * deltaUV21 - deltaUV11 * deltaUV20) );
    }
    let tangentX = (deltaPos10 * deltaUV21 - deltaPos20 * deltaUV11)*r;
    let tangentY = (deltaPos11 * deltaUV21 - deltaPos21 * deltaUV11)*r;
    let tangentZ = (deltaPos12 * deltaUV21 - deltaPos22 * deltaUV11)*r;
    for( let j = 0; j < 3; j++ ){
      tagents.push(tangentX);
      tagents.push(tangentY);
      tagents.push(tangentZ);
    }
    let bitangentX = (deltaPos20 * deltaUV10 - deltaPos10 * deltaUV20)*r;
    let bitangentY = (deltaPos21 * deltaUV10 - deltaPos11 * deltaUV20)*r;
    let bitangentZ = (deltaPos22 * deltaUV10 - deltaPos12 * deltaUV20)*r;
    for( let j = 0; j < 3; j++ ){
      bitagents.push(bitangentX);
      bitagents.push(bitangentY);
      bitagents.push(bitangentZ);
    }
  }
  let obj = {};
  obj['tagents'] = tagents;
  obj['bitagents'] = bitagents;
  obj['crossTexCoords'] = crossTexCoords;
  return obj;
```

Example (Ex13-3)

- loadOBJtoCreateVBO() and drawOneRegularObject() in WebGL.js
  - So, we have to create VBO for the cross products array and pass then into the shader before drawing the object
  - Many corresponding self-defined functions should be changed, we do not go through all the details here

```
async function loadOBJtoCreateVBO( objFile ){
  let objComponents = [];
  response = await fetch(objFile);
  text = await response.text();
  obj = parseOBJ(text);
  for( let i=0; i < obj.geometries.length; i ++ ){
    let tagentSpace = calculateTangentSpace(obj.geometries[i].data.position,
                                            obj.geometries[i].data.texcoord);
    let o = initVertexBufferForLaterUse(gl,
                                        obj.geometries[i].data.position,
                                        obj.geometries[i].data.normal,
                                        obj.geometries[i].data.texcoord,
                                        tagentSpace.tagents,
                                        tagentSpace.bitagents,
                                        tagentSpace.crossTexCoords);
    objComponents.push(o);
  }
  return objComponents;
}
```

```
function drawOneRegularObject(obj, modelMatrix, vpMatrix, colorR, colorG, colorB){
  gl.useProgram(program);
  let mvpMatrix = new Matrix4();
  let normalMatrix = new Matrix4();
  mvpMatrix.set(vpMatrix);
  mvpMatrix.multiply(modelMatrix);

  //normal matrix
  normalMatrix.setInverseOf(modelMatrix);
  normalMatrix.transpose();

  gl.uniform3f(program.u_LightPosition, lightX, lightY, lightZ);
  gl.uniform3f(program.u_ViewPosition, cameraX, cameraY, cameraZ);
  gl.uniform1f(program.u_Ka, 0.2);
  gl.uniform1f(program.u_Kd, 0.7);
  gl.uniform1f(program.u_Ks, 1.0);
  gl.uniform1f(program.u_shininess, 40.0);
  gl.uniform3f(program.u_Color, colorR, colorG, colorB);
  gl.uniform1i(program.u_Sampler0, 0);
  gl.uniform1i(program.u_Sampler1, 1);

  gl.uniformMatrix4fv(program.u_MvpMatrix, false, mvpMatrix.elements);
  gl.uniformMatrix4fv(program.u_modelMatrix, false, modelMatrix.elements);
  gl.uniformMatrix4fv(program.u_normalMatrix, false, normalMatrix.elements);

  gl.activeTexture(gl.TEXTURE0);
  gl.bindTexture(gl.TEXTURE_2D, textures["normalMapImage"]);

  for( let i=0; i < obj.length; i ++ ){
    initAttributeVariable(gl, program.a_Position, obj[i].vertexBuffer);
    initAttributeVariable(gl, program.a_Normal, obj[i].normalBuffer);
    initAttributeVariable(gl, program.a_TexCoord, obj[i].texCoordBuffer);
    initAttributeVariable(gl, program.a_Tagent, obj[i].tagentsBuffer);
    initAttributeVariable(gl, program.a_Bitagent, obj[i].bitagentsBuffer);
    initAttributeVariable(gl, program.a_crossTexCoord, obj[i].crossTexCoordsBuffer);
    gl.drawArrays(gl.TRIANGLES, 0, obj[i].numVertices);
  }
}
```

Example (Ex13-3)

- Vertex shader in WebGL.js
  - Before making the TBN matrix, we check a_crossTexCoord to determine we should use $T \times B$ or $B \times T$ to calculate N

- Fragment shader is the same as Ex13-2

```
var VSHADER_SOURCE = `
    attribute vec4 a_Position;
    attribute vec2 a_TexCoord;
    attribute vec4 a_Normal;
    attribute vec3 a_Tagent;
    attribute vec3 a_Bitagent;
    attribute float a_crossTexCoord;
    uniform mat4 u_MvpMatrix;
    uniform mat4 u_modelMatrix;
    uniform mat4 u_normalMatrix;
    varying vec3 v_PositionInWorld;
    varying vec2 v_TexCoord;
    varying mat4 v_TBN;
    varying vec3 v_Normal;
    void main(){
        gl_Position = u_MvpMatrix * a_Position;
        v_PositionInWorld = (u_modelMatrix * a_Position).xyz;
        v_Normal = normalize(vec3(u_normalMatrix * a_Normal));
        v_TexCoord = a_TexCoord;
        //create TBN matrix
        vec3 tagent = normalize(a_Tagent);
        vec3 bitagent = normalize(a_Bitagent);
        vec3 nVector;
        if( a_crossTexCoord > 0.0){
          nVector = cross(tagent, bitagent);
        } else{
          nVector = cross(bitagent, tagent);
        }
        v_TBN = mat4(tagent.x, tagent.y, tagent.z, 0.0,
                          bitagent.x, bitagent.y, bitagent.z, 0.0,
                          nVector.x, nVector.y, nVector.z, 0.0,
                          0.0, 0.0, 0.0, 1.0);
    }
`;
```