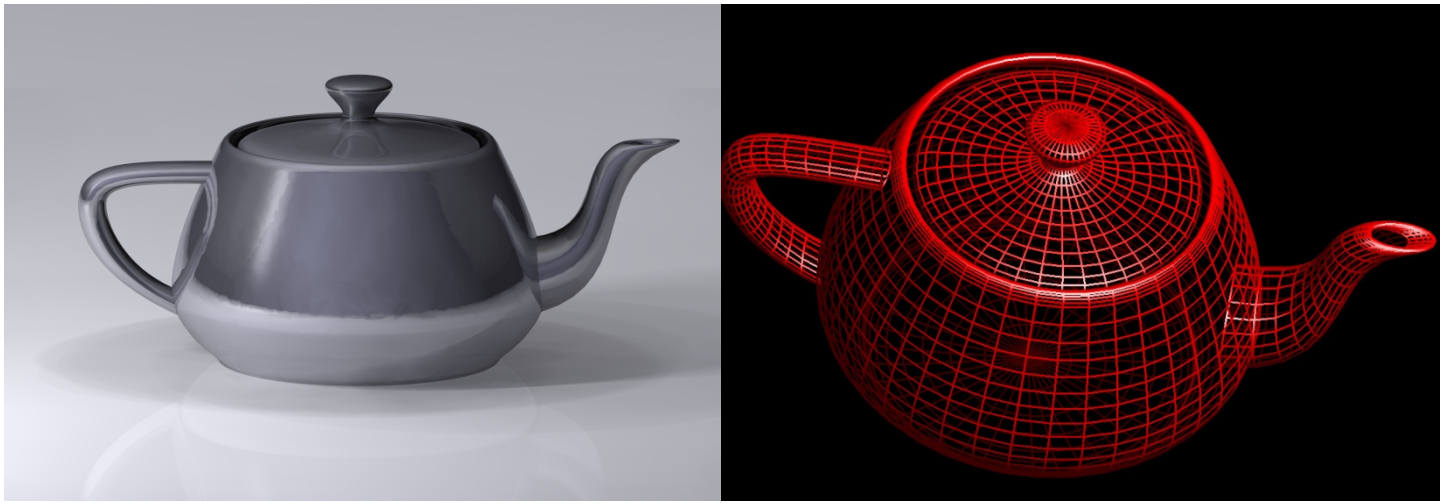


Foundations of Computer Graphics

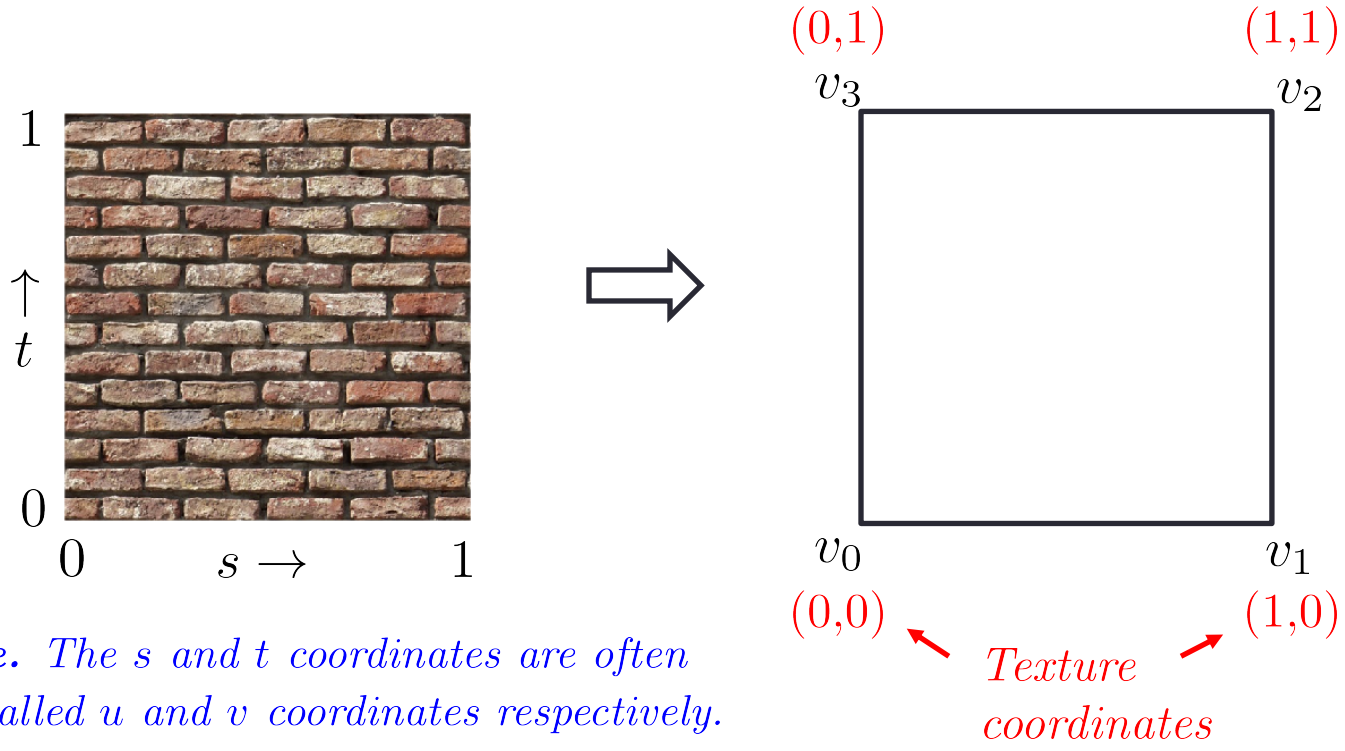
Saurabh Ray

We will start in a few minutes.

Tuesday, April 7 instead of April 2.

Texture Mapping

Minimal example. *Pasting a picture on a square.*



Note. *The s and t coordinates are often also called u and v coordinates respectively.*

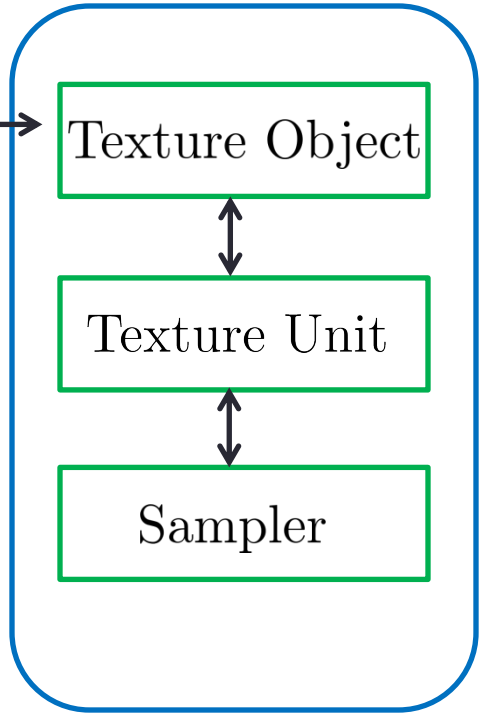
We need:

- positions of the vertices
- texture coordinates of the vertices

Texture Mapping

Image object

create an image object in Javascript.



create a WebGL texture object

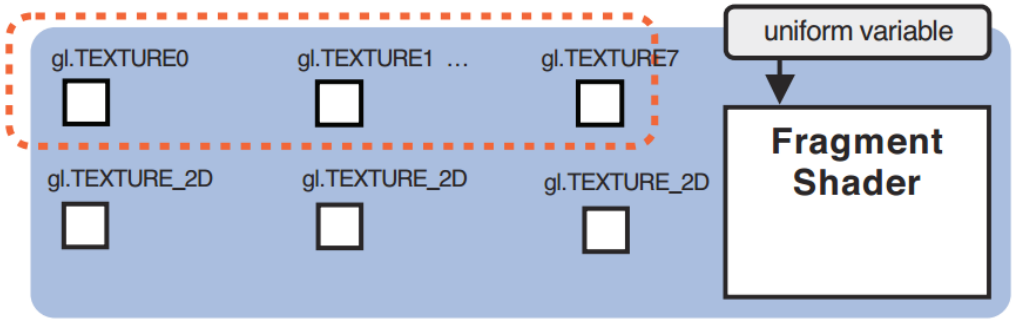
associate the image with it

associate the texture object with a texture unit

create a sampler in the shader and associate it with the texture unit

There are multiple texture units, usually at least 8.

We can create multiple texture objects and assign them to different texture units.



Texture Mapping

```
"use strict";
var gl; // global variable
var image;

window.onload = function init(){
    //Set  up WebGL
    var canvas = document.getElementById( "gl-canvas" );
    gl = WebGLUtils.setupWebGL( canvas );
    if ( !gl ) {alert( "WebGL isn't available" );}

    // Set viewport and clear canvas
    gl.viewport( 0, 0, canvas.width, canvas.height );
    gl.clearColor( 0.0, 0.0, 0.0, 1.0 );
    gl.clear(gl.COLOR_BUFFER_BIT);

    // Load shaders and initialize attribute buffers
    var program = initShaders( gl, "vertex-shader", "fragment-shader" );
    gl.useProgram( program );
```

Texture Mapping

```
// Set up buffers and attributes
var s = 0.7;
var vertices =  [-s,-s,  s,-s,  s,s,  -s,s];
var texCoords = [ 0,0,   1,0,  1,1,    0,1];
var indices =   [0,1,2,  0,2,3];

var vbuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vbuffer);
gl.bufferData(gl.ARRAY_BUFFER, flatten(vertices), gl.STATIC_DRAW);
var vPosition = gl.getAttribLocation(program, "vPosition");
gl.vertexAttribPointer(vPosition, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vPosition);
```

Associating texture coordinates with vertices.


```
var tbuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, tbuffer);
gl.bufferData(gl.ARRAY_BUFFER, flatten(texCoords), gl.STATIC_DRAW);
var vTexCoord = gl.getAttribLocation(program, "vTexCoord");
gl.vertexAttribPointer(vTexCoord, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(vTexCoord);
```

```
var ibuffer = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, ibuffer);
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER, new Uint8Array(flatten(indices)), gl.STATIC_DRAW);
```

Texture Mapping

```
    var texture = gl.createTexture();
    var mySampler = gl.getUniformLocation(program, "mySampler");

    image = new Image();
    image.onload = function(){handler(texture);};
    image.src = "brick.gif";

    function handler(texture){
        gl.activeTexture(gl.TEXTURE0);            // enable texture unit 0
        gl.bindTexture(gl.TEXTURE_2D, texture);  // bind texture object to target
        gl.uniform1i(mySampler, 0);              // connect sampler to texture unit 0

        gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true); // flip image's y axis
        gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, image);
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
        gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.NEAREST);

        gl.drawElements(gl.TRIANGLES,6,gl.UNSIGNED_BYTE,0);

    }
};
```

We do three things:

1. Create a texture object and associate it with an image.
2. Associate a sampler with the texture object.
3. Associate the sampler with a texture unit.

Texture Mapping

```
<script id="vertex-shader" type="x-shader/x-vertex">
precision mediump float;

attribute vec4 vPosition;
attribute vec2 vTexCoord;

varying vec2 fTexCoord;

void main(){
    gl_Position = vPosition;
    fTexCoord = vTexCoord;
}
</script>
```

Using a varying variable to get texture coordinates at each fragment.

```
<script id="fragment-shader" type="x-shader/x-fragment">
precision mediump float;

varying vec2 fTexCoord;

uniform sampler2D mySampler;

void main(){
    gl_FragColor = texture2D(mySampler,fTexCoord);
}
</script>
```

Using the sampler.

Multiple Textures

Example:

create two texture objects

```
texture1 = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, texture1);
gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, texSize, texSize, 0,
              gl.RGBA, gl.UNSIGNED_BYTE, image1);
gl.generateMipmap(gl.TEXTURE_2D);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER,
gl.NEAREST_MIPMAP_LINEAR);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);

texture2 = gl.createTexture();
gl.bindTexture(gl.TEXTURE_2D, texture2);
gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);
gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, texSize, texSize, 0,
              gl.RGBA, gl.UNSIGNED_BYTE, image2);
gl.generateMipmap(gl.TEXTURE_2D);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER,
gl.NEAREST_MIPMAP_LINEAR);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.NEAREST);
```

Multiple Textures

In the fragment shader, we need two samplers to access them.

```
uniform sampler2D Tex0;
uniform sampler2D Tex1;
```

create two samplers

Connect the texture objects with samplers.

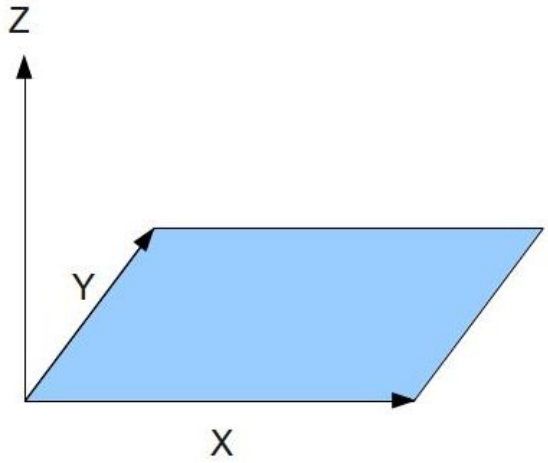
```
gl.activeTexture(gl.TEXTURE0);      subsequent calls affect texture unit 0
gl.bindTexture(gl.TEXTURE_2D, texture1);
gl.uniform1i(gl.getUniformLocation(program, "Tex0"), 0);

gl.activeTexture(gl.TEXTURE1);      subsequent calls affect texture unit 1
gl.bindTexture(gl.TEXTURE_2D, texture2);
gl.uniform1i(gl.getUniformLocation(program, "Tex1"), 1);
```

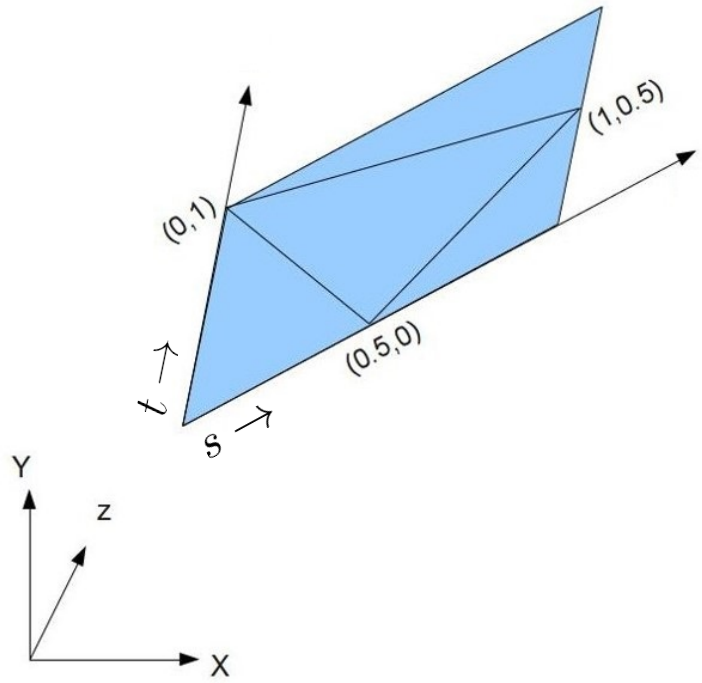
Now the fragment shader can do something like this:

```
gl_FragColor = color * texture2D(Tex0, texCoord)
                     * texture2D(Tex1, texCoord);
```

Normal Mapping



Coordinate system in which normals are defined.

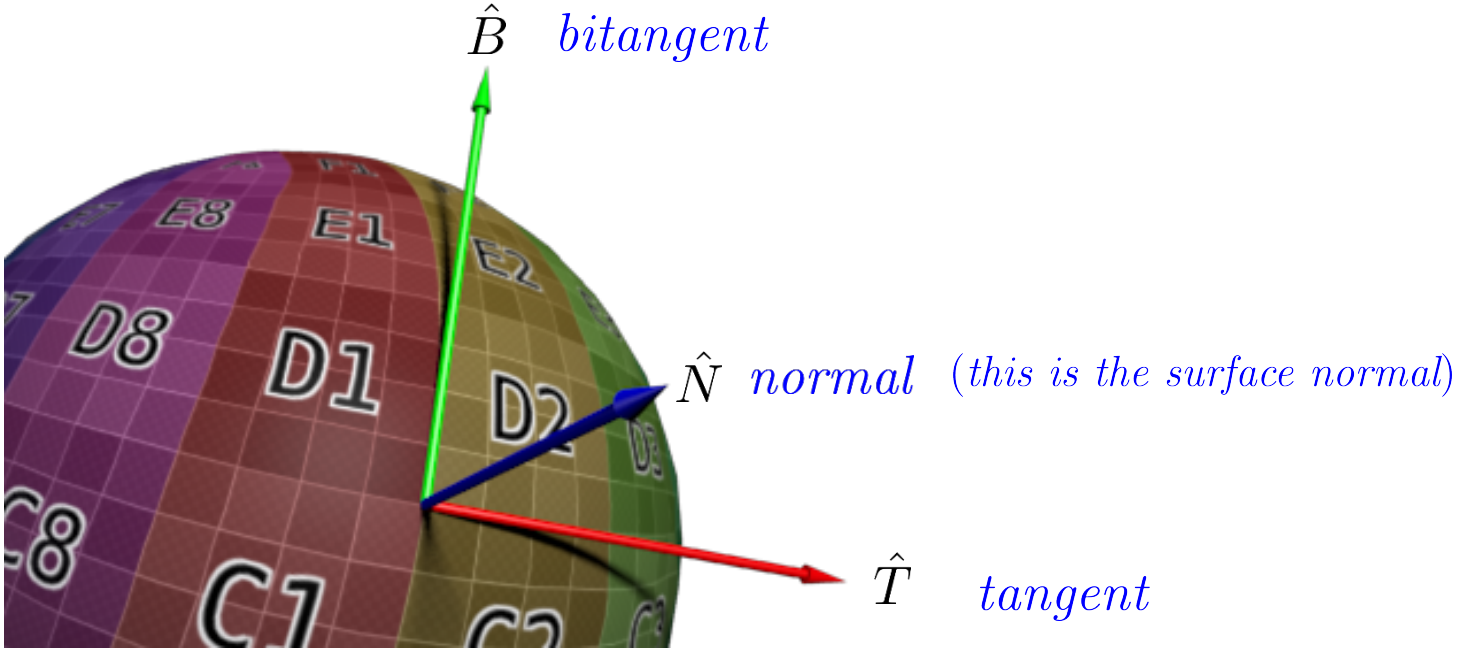


Placing the normal map on a triangle whose vertices have texture coordinates $(0.5, 0)$, $(1, 0.5)$ *and* $(0, 1)$.

The normal we get from the texture needs to be modified according to the orientation of surface.

Normal Mapping

In general, we use a "local" coordinate frame to interpret the coordinates.



The "modified" normal we get from the texture is described in this frame.

We convert it to the coordinate frame where we do lighting.

Normal Mapping

The tangent frame can be easily computed for surfaces described by an equation using partial derivatives.

How do we compute the tangent frame for triangular meshes?

- For each triangle, compute the tangent, bitangent and normal

- For each vertex, average the values at adjacent triangles

- For fragments, we get the vectors by linear interpolation

Note: This leads to slight inaccuracy since after the averaging and interpolation, the three vectors are not perpendicular to each other.

We ignore this since if the triangles are small, the error is small.

Normal Mapping

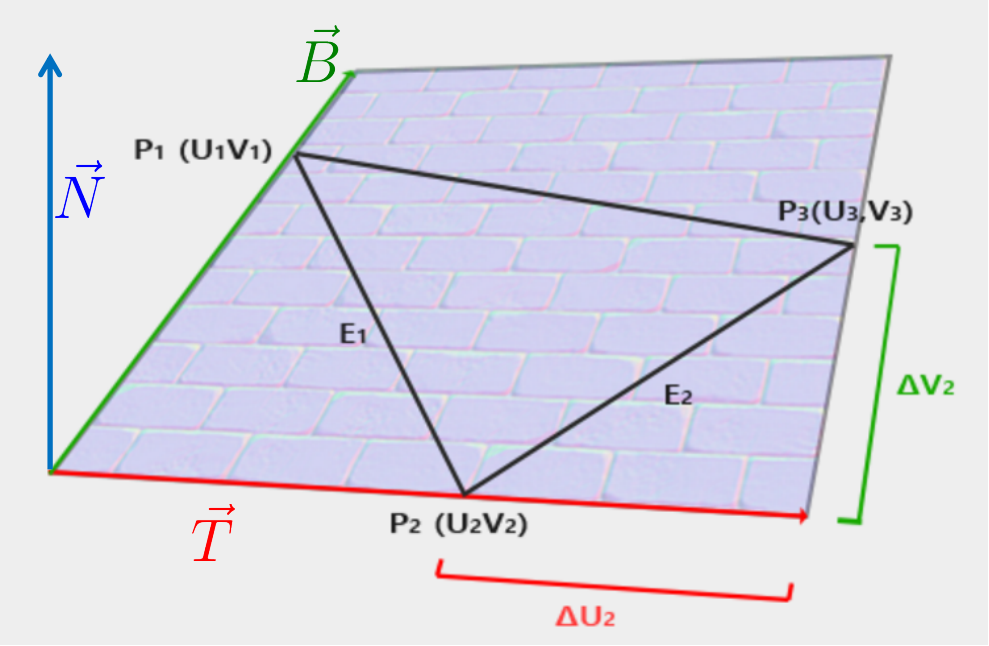
How do we compute tangents and bitangent on a triangle?

We have texture coordinates at each vertex.

These are along $\vec{T}$ and $\vec{B}$.

We also know the positions of the vertices.

From this we can figure out $\vec{T}$ and $\vec{B}$.



This is how it would look if we put the entire texture in the plane of the triangle so that the texture coordinates at the vertices match.

Note that $\vec{T}$ and $\vec{B}$ need not be of unit length.

They are as long as the length and height of the texture respectively.

Normal Mapping

Let $\vec{E}_1 = P_2 - P_1$

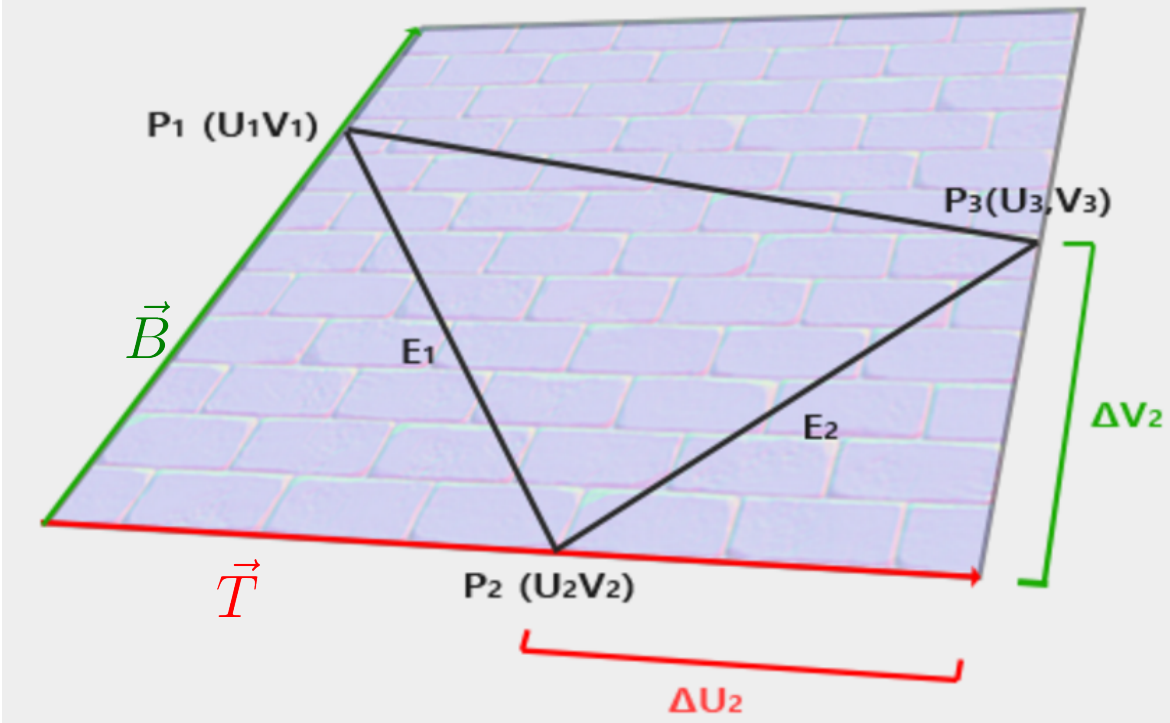
$\vec{E}_2 = P_3 - P_2$

$\Delta U_1 = U_2 - U_1$

$\Delta V_1 = V_2 - V_1$

$\Delta U_2 = U_3 - U_2$

$\Delta V_2 = V_3 - V_2$



$\vec{E}_1 = \Delta U_1 \vec{T} + \Delta V_1 \vec{B}$

$\vec{E}_2 = \Delta U_2 \vec{T} + \Delta V_2 \vec{B}$

two equations in two variables

Normal Mapping

$$\vec{E}_1 = \Delta U_1 \vec{T} + \Delta V_1 \vec{B}$$

$$\vec{E}_2 = \Delta U_2 \vec{T} + \Delta V_2 \vec{B}$$

Expanding to three coordinates:

$$(E_{1x}, E_{1y}, E_{1z}) = \Delta U_1 (T_x, T_y, T_z) + \Delta V_1 (B_x, B_y, B_z)$$

$$(E_{2x}, E_{2y}, E_{2z}) = \Delta U_2 (T_x, T_y, T_z) + \Delta V_2 (B_x, B_y, B_z)$$

In matrix notation:

$$\begin{bmatrix} E_{1x} & E_{1y} & E_{1z} \\ E_{2x} & E_{2y} & E_{2z} \end{bmatrix} = \begin{bmatrix} \Delta U_1 & \Delta V_1 \\ \Delta U_2 & \Delta V_2 \end{bmatrix} \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \end{bmatrix}$$

Normal Mapping

$$\begin{bmatrix} E_{1x} & E_{1y} & E_{1z} \\ E_{2x} & E_{2y} & E_{2z} \end{bmatrix} = \begin{bmatrix} \Delta U_1 & \Delta V_1 \\ \Delta U_2 & \Delta V_2 \end{bmatrix} \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} \Delta U_1 & \Delta V_1 \\ \Delta U_2 & \Delta V_2 \end{bmatrix}^{-1} \begin{bmatrix} E_{1x} & E_{1y} & E_{1z} \\ E_{2x} & E_{2y} & E_{2z} \end{bmatrix} = \begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \end{bmatrix}$$

$$\Downarrow$$

$$\begin{bmatrix} T_x & T_y & T_z \\ B_x & B_y & B_z \end{bmatrix} = \frac{1}{\Delta U_1 \Delta V_2 - \Delta U_2 \Delta V_1} \begin{bmatrix} \Delta V_2 & -\Delta V_1 \\ -\Delta U_2 & \Delta U_1 \end{bmatrix} \begin{bmatrix} E_{1x} & E_{1y} & E_{1z} \\ E_{2x} & E_{2y} & E_{2z} \end{bmatrix}$$

Normal Mapping

We now need two more vertex attributes: **tangent** and **bitangent**.

Just like normals, these are interpolated for each fragment so that we get the normal, tangent and bitangent at each fragment.

Now, we use the texture coordinates to read the color (r, g, b) from the normal map texture.

Convert color to normal in the tangent frame: $(2r-1, 2g-1, 2b-1)$

This is converted to world frame for lighting:

$$\vec{N}' = (2r-1)\hat{T} + (2g-1)\hat{B} + (2b-1)\hat{N}$$

where $\hat{T} = \frac{\vec{T}}{\|\vec{T}\|}$, $\hat{B} = \frac{\vec{B}}{\|\vec{B}\|}$ and $\hat{N} = \frac{\vec{N}}{\|\vec{N}\|}$.