Triangular Mesh Compression

topologic approach

Nicolas Bloyet - https://github.com/theplatypus

12/07/2016

- I - Abstract
- II – Modules
- III – Topologic compression
- IV – Stripes displaying

I - Abstract

Leverage the mesh topology to an additional compression step

.obj format

The faces of our mesh (triangles) are defined by the respective index of their inner elements.

⇒ Each triangle is described individually

⇒ Does not mention common sides

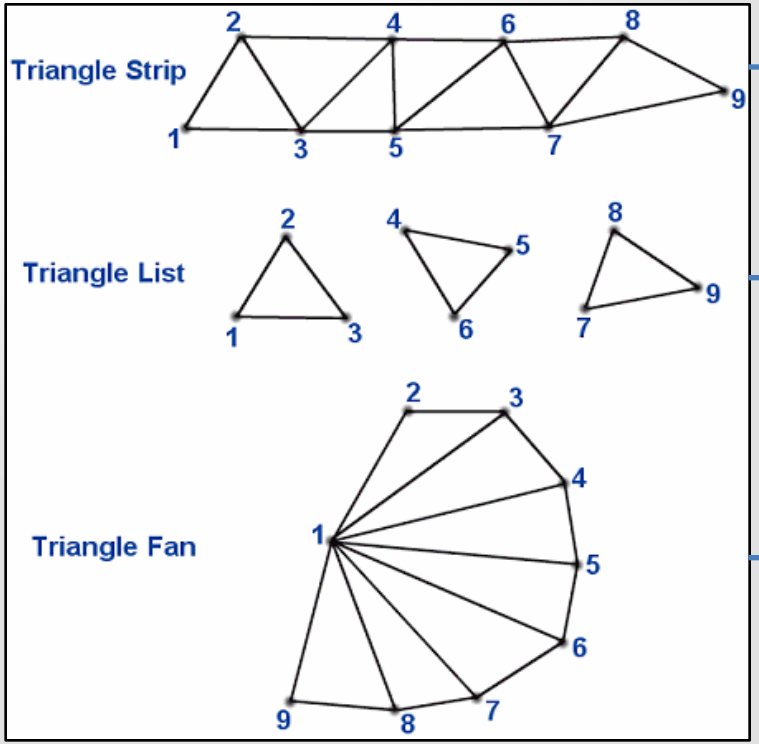
This information redundancy could be an additional compression step (before dictionnary compression)

```
# Sample of .obj file

# List of geometric vertices, with (x,y,z)
v 0.123 0.234 0.345
v ... ...

# List of texture coordinates, in (u, v [,w])
vt 0.500 1 [0]
vt ... ...

# List of vertex normals in (x,y,z)
vn 0.707 0.000 0.707
vn ... ...

# Polygonal face elements
# f v1/vt1/vn1 v2/vt2/vn2 v3/vt3/vn3
f 1// 2// 3//
f 3// 4// 5//
f 5// 3// 7//
f ... ...
```

Abstract

Several ways to describe a triangle's set

Let be 9 vertices (n)…

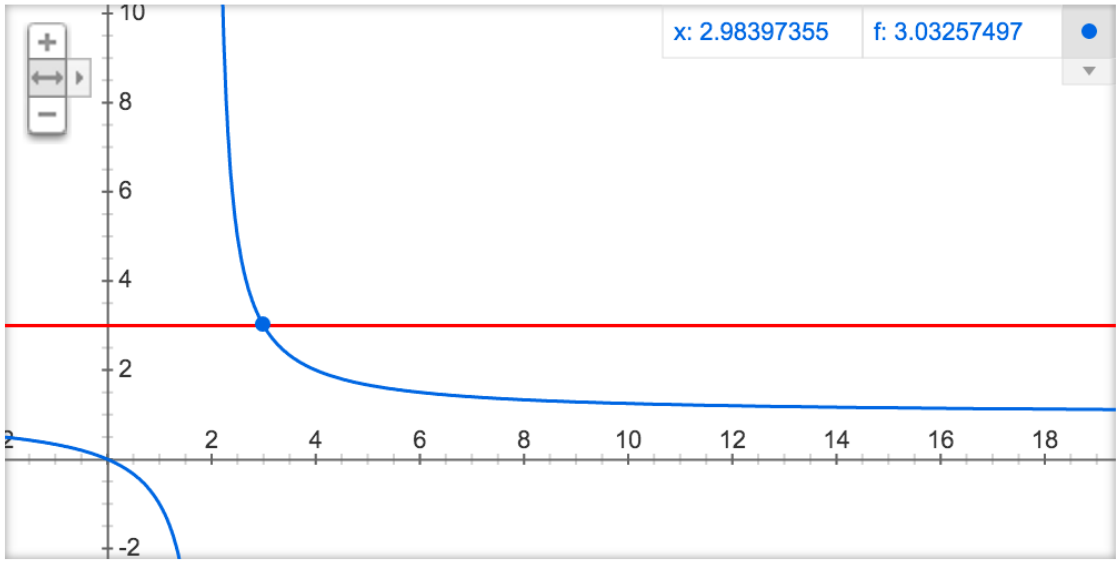


7 triangles (n-2)
vertices / triangle : 1.28
= (n / n-2)

3 triangles (n/3)
vertices / triangle : 3
= (n / (n/3))
= 3 $\forall$ n

7 triangles (n-2)
vertices / triangle : 1.28
= (n / n-2)

…but fan is less flexible than strip

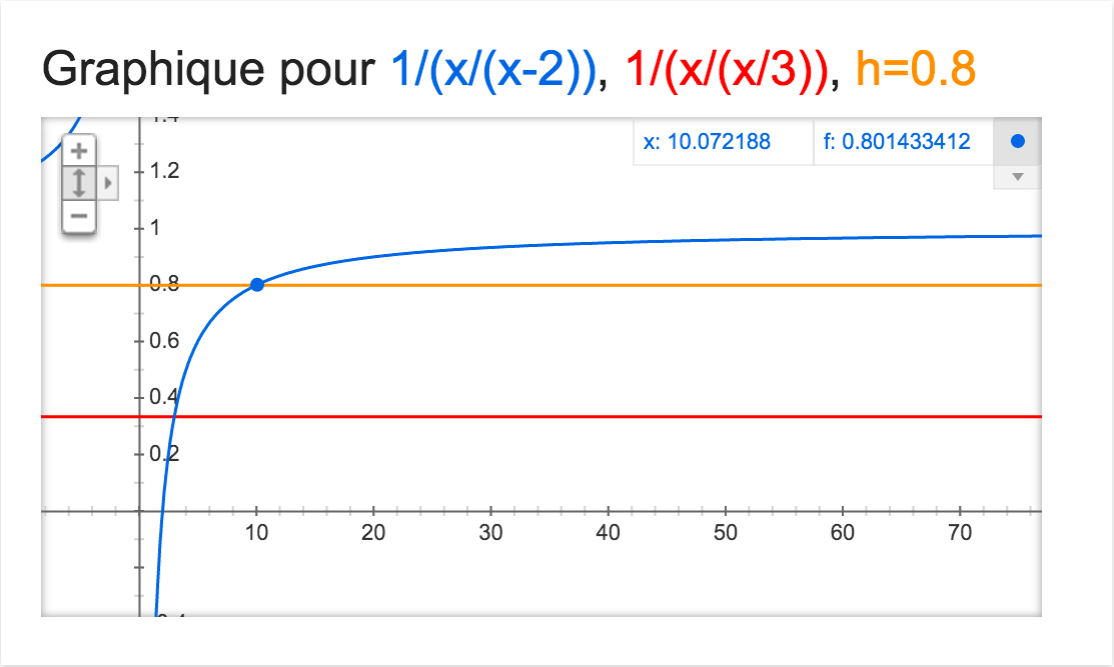


- Equal in the worst case
- More efficient as n grow

=> Strips are at least as efficient as independent triangles

Abstract
Information amount contained in a vertice

Drawing the inverse function of previous ones, we obtain the number of faces described by each vertice, which means the « information quantity » contained by each vertice.



- 80 % of optimal efficiency (asymptotic) reached when n = 10

=> A 10 triangle's length strip is quite common in « real life » meshes, so this approach is interesting !

II – Modules structure

OpenGL ES 2 (specification) implementation, OpenGL's subset

+

- comparable performances to OpenGL
- only require a compatible browser (cross-platform)
- allows the use of GPU
- very good integration in WebApp
- Interface design easy (HTML/CSS)
- applicative code in JavaScript (which is quite easy to learn and hack)

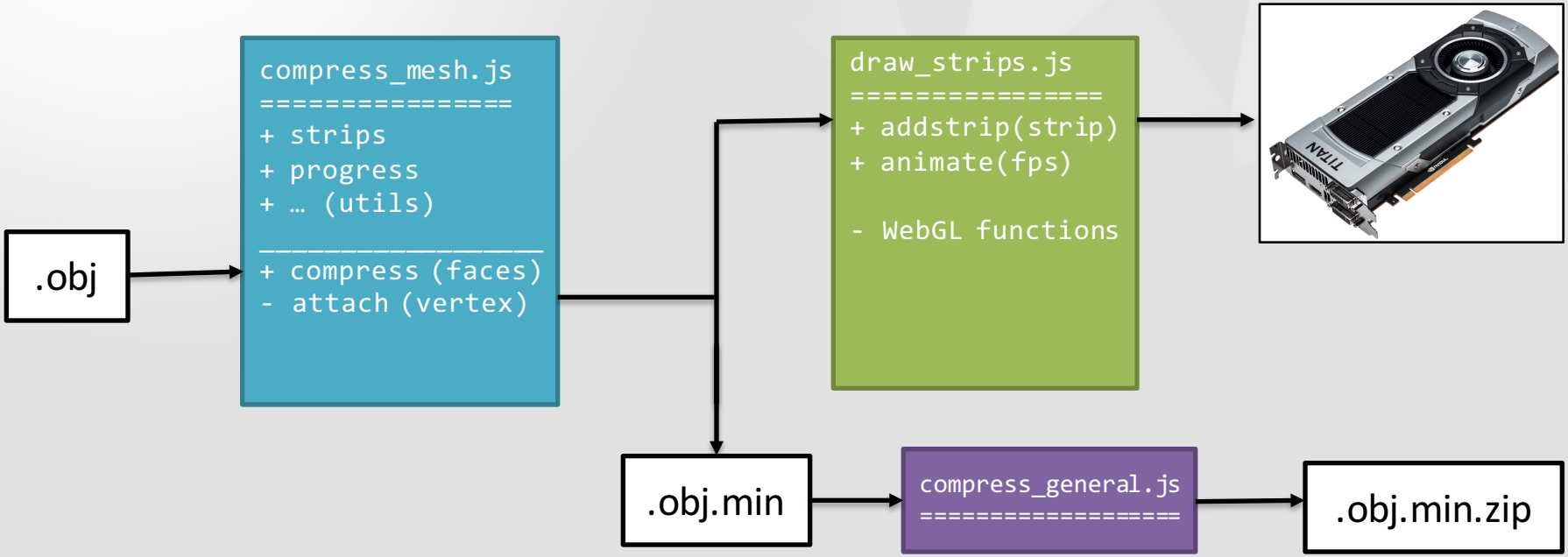
OpenGL ES 2 (specification) implementation, OpenGL's subset

-

- Currently less functionnalities than desktop OpenGL
(you have to write shaders to use light, etc.)

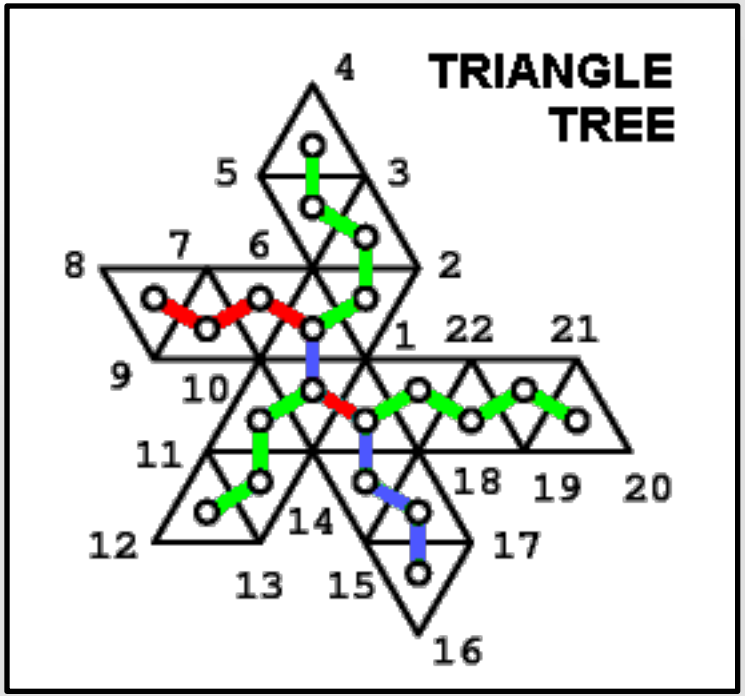
Technology chosen by curiosity first, and affinity with JavaScript and web development in general on the other hand.

Modules


.obj
compress_mesh.js
+ strips
+ progress
+ … (utils)
+ compress (faces)
- attach (vertex)
draw_strips.js
+ addstrip(strip)
+ animate(fps)
- WebGL functions
TITAN
.obj.min
compress_general.js
.obj.min.zip

III – Topologic Compression

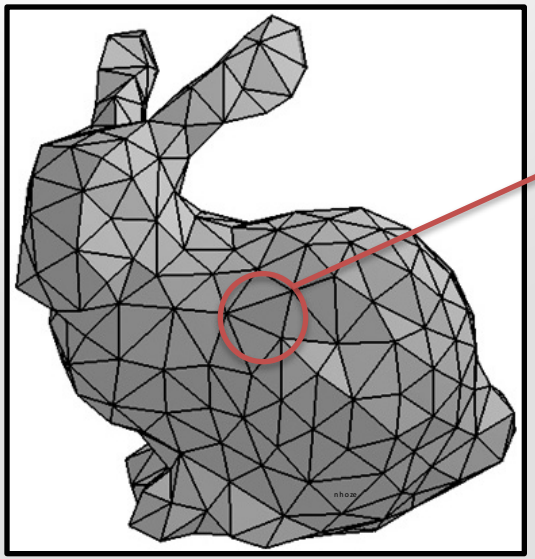
Optimal algorithm



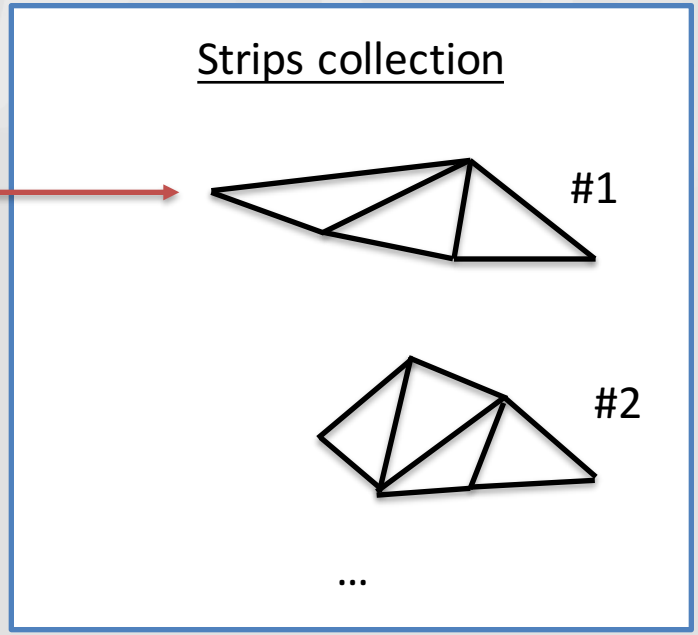
⇒ Minimum Spanning Tree (well-known problem)

⇒ But not so easy to code…

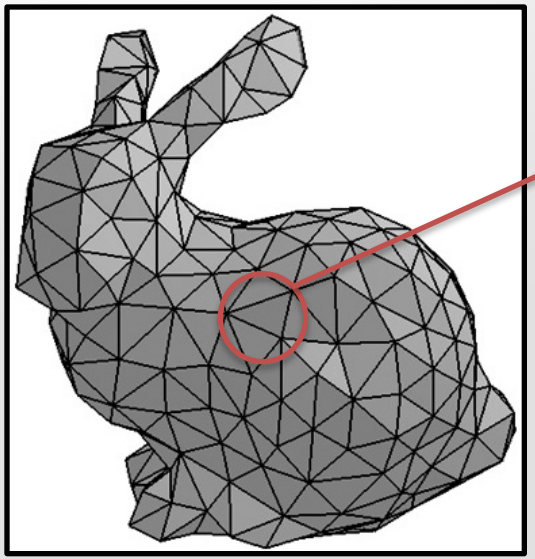
Naive algorithm (but quite effective)



is the side completes the strip ?

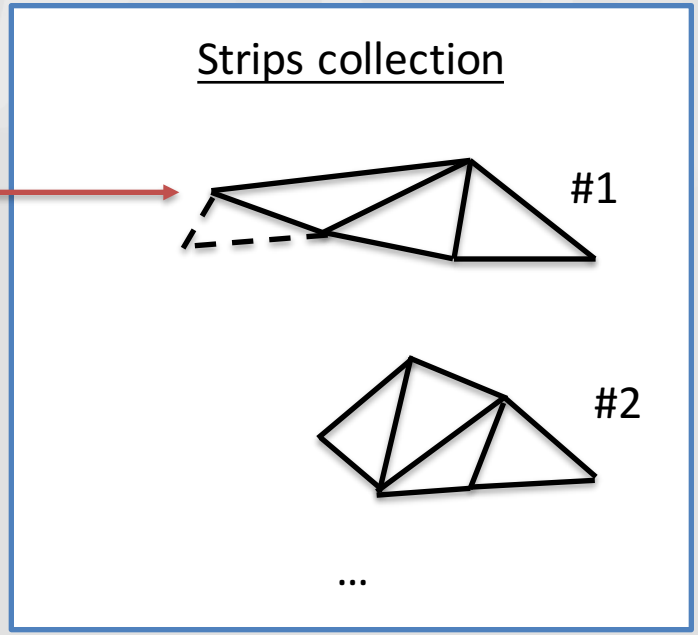


Naive algorithm (but quite effective)

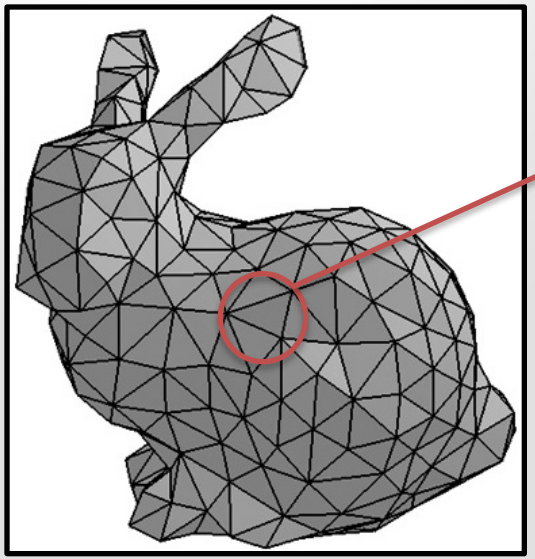


is the side completes the strip ?

if yes…

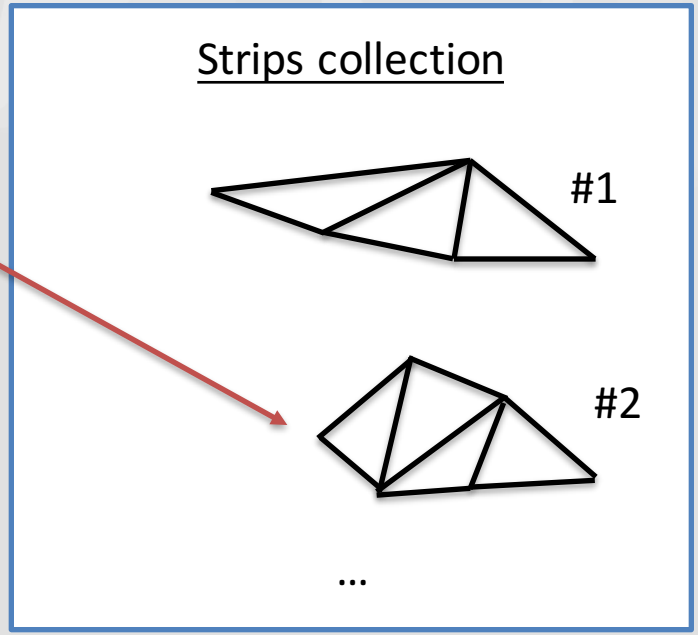


Naive algorithm (but quite effective)



is the side completes the strip ?

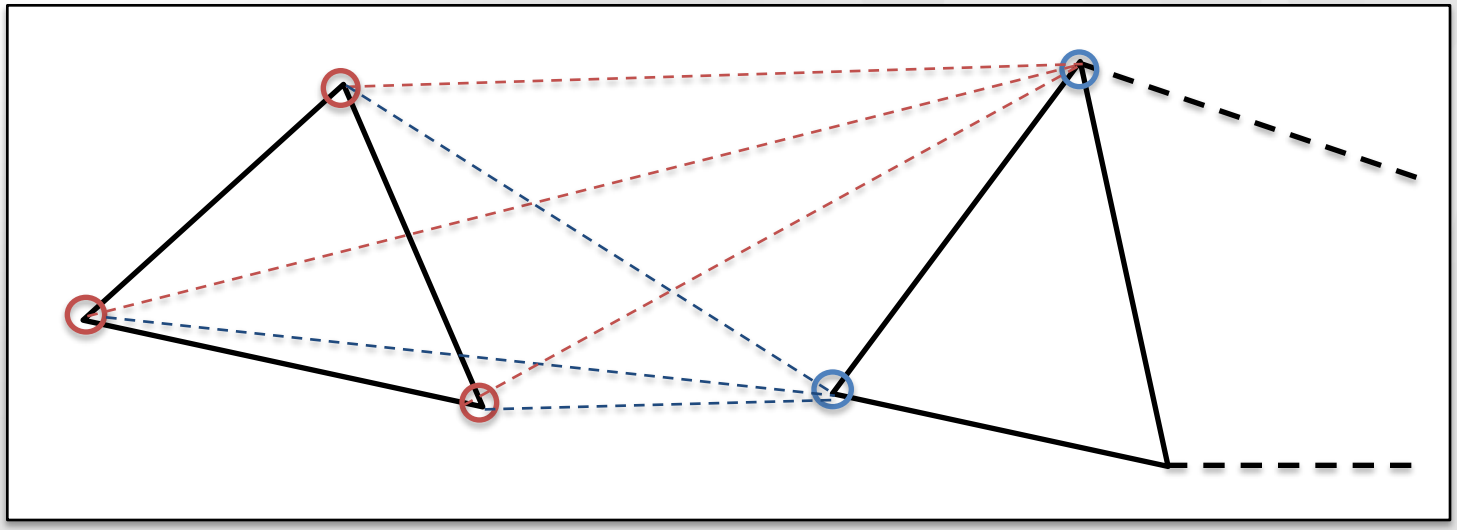
else…



etc.

no matching => new strip

Continuity test



A side completes a strip if this side and strip's end have exactly two vertices in common.

Continuity test

Let be :

- a side F : [f1, f2, f3]
- an end E : [e1, e2]

If f1 ∈ E
    if (f2 ∈ E) return f3
    else if (f3 ∈ E) return f2
    else Fail
Else if (f2 ∈ E)
    if (f1 ∈ E) return f3
    else if (f3 ∈ E) return f1
    else Fail
Else Fail

```
/**
 *  Test wether a face could be attached to an existent strip's end
 *
 *  @param vertices Array[3 * int] candidate face
 *  @param end      Array[2 * int] end of strip
 *  @return null | point to add
 */
function attach(vertices, end){

    let v1 = vertices[0],
        v2 = vertices[1],
        v3 = vertices[2];

    if(end.indexOf(v1) !== -1){
        if(end.indexOf(v2) !== -1) return v3 ;
        else if(end.indexOf(v3) !== -1) return v2 ;
        else return null ;
    }else if(end.indexOf(v2) !== -1){
        if(end.indexOf(v1) !== -1) return v3 ;
        else if(end.indexOf(v3) !== -1) return v1 ;
        else return null ;
    }else return null;
}
```

Complexity

Depends on comparison level (wether we test the front and/or the strip's end)
With n sides :

- simple comparison : $O(n^2)$
- double comparison : $O(2 * n^2)$

=> quite heavy…

IV – Strips animation

Assign an OpenGL buffer for each strip

For each strip obtained in compression step, we get an Array[Integer].

- Each of those arrays is copied in an static OpenGL buffer

  (because we do not edit sides displaying them)

- We store each strip's length

```
/**
 *   addStrip
 *   ========
 *
 *   Add a strip to display. Strips may have differents length.
 *   Indexes are integers.
 *   @param {Array} stripVertexIndices : an Array of Indexes describing a strip
 */
draw_strips.addStrip = function(stripVertexIndices) {

    // Build the element array buffer; this specifies the indices
    // into the vertex array for each face's vertices.

    stripVerticesIndexBuffer[nb_strips] = gl.createBuffer();
    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, stripVerticesIndexBuffer[nb_strips]);

    // Now send the element array to GL
    var int16 = new Uint16Array(stripVertexIndices);
    gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
        int16, gl.STATIC_DRAW);

    t_max[nb_strips] = int16.length;

    nb_strips++;
}
```

Animate on strip's length

```
/**
 *   animate
 *   =======
 *
 *   Display progessively all the strips registered at this time
 *   @param {Integer} fps : frames per second
 */
draw_strips.animate = function(fps) {

    t = 0;
    freq = (1000 / fps);
    setInterval(function() {

        drawScene();

        var currentTime = (new Date).getTime();

        if (lastStripUpdateTime) {
            var delta = currentTime - lastStripUpdateTime;
            if (draw_strips.autoX) draw_strips.rotationHorizontal += (draw_strips.autoXvalue * delta) / 1000.0;
            if (draw_strips.autoY) draw_strips.rotationVertical += (draw_strips.autoYvalue * delta) / 1000.0;
            if (draw_strips.autoZ) draw_strips.rotationZ += (draw_strips.autoZvalue * delta) / 1000.0;
        }
        lastStripUpdateTime = currentTime;
        if (!draw_strips.pauseAnimation){
            t++;
            draw_strips.progress = (t / draw_strips.speedAnimation);
        }
    }, freq);
}
```

Display each strip

- use of primitive
  TRIANGLE_STRIP
- strips drawing, narrowed on animation progression

```
// Draw the strips.
for (var n = 0; n <= nb_strips; n++) {

    gl.bindBuffer(gl.ARRAY_BUFFER, stripVerticesBuffer);
    gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);

    // Set the texture coordinates attribute for the vertices.

    gl.bindBuffer(gl.ARRAY_BUFFER, stripVerticesTextureCoordBuffer);
    gl.vertexAttribPointer(textureCoordAttribute, 2, gl.FLOAT, false, 0, 0);

    // Bind the normals buffer to the shader attribute.

    gl.bindBuffer(gl.ARRAY_BUFFER, stripVerticesNormalBuffer);
    gl.vertexAttribPointer(vertexNormalAttribute, 3, gl.FLOAT, false, 0, 0);

    // Specify the texture to map onto the faces.

    gl.activeTexture(gl.TEXTURE0);
    gl.bindTexture(gl.TEXTURE_2D, stripTexture);
    gl.uniform1i(gl.getUniformLocation(shaderProgram, "uSampler"), 0);

    gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, stripVerticesIndexBuffer[n]);
    setMatrixUniforms();
    var strip_length = Math.min(draw_strips.progress, t_max[n]);
    gl.drawElements(gl.TRIANGLE_STRIP, strip_length, gl.UNSIGNED_SHORT, 0);

};
```

Thaks for reading

☺