Digital Geralt report

Malo Malcom Drougard, Jad Khoury and Augustin Prado

July 1, 2017

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Abstract

For this project we were given a 3D model of a lamp that is 3D-printable and a scanned mesh (the Geralt mesh). We were asked to build from the two above-mentionned objects a new mesh. The goal of this work was to modify the Geralt head such that it becomes a lampshade compatible with the provided lamp model. Our team approach was to sculpt the mesh, as procedurally as possible, such that it looks like it's being rendered. To achieve this effect, we played with remeshing and created a custom algorithm to convert the surface into a wireframe-looking solid. This report comes to enlighten all the steps we followed from the beginning of the project and the difficulties we encountered. For curious readers, our code is available at our github repository www.github.com/jadkhoury/DigitalGeralt- .

Introduction

During the development of the project and the creation of the final mesh, that we will call the "digital geralt", we decided to change slightly our first pipeline by adding more steps, that we describe below.

Finally, the Digital Geralt is a composition of many different methods :

1. **Mesh Preprocessing:** Remove the long hair from the head of the Geralt mesh and cut the neck.
2. **Directional Remeshing:** Modify the heiht-base remeshing algorithm seen in class such that it can scale the target-length not only in function of the height, but following of any direction
3. **Wireframe:** Change a part of the mesh to a wireframe shape
4. **Flowerize:** Try some other "wireframe" patterns
5. **Thickness:** Create and run an algorithm aiming at converting the surface into a solid
6. **Support:** Create a cylinder support for the lamp

1 Mesh Preprocessing

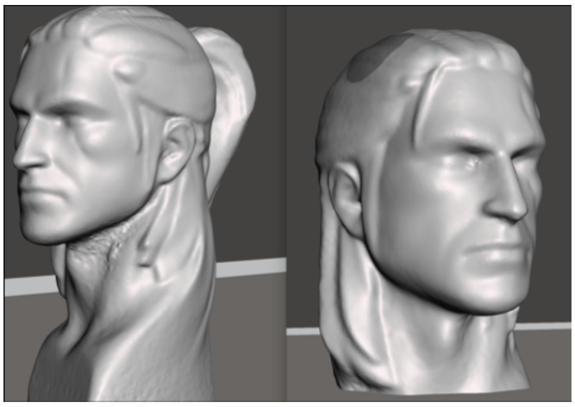

Figure 1: Before and after pre-processing.

We used MeshMixer to remove the ponytail of Geralt that was in the way of our objective. To do so, we used the select tool Discard on the big part of the ponytail to delete the edges and vertices, then we used the edit tool select tool Erase and Fill to fill the holes left by the haircut. After Geralt's haircut was done, we smoothed the part of the mesh that was suffering from acnea caused by the high-poly topology of the scan and its lack of precision in occluded places (under the chin for example, as you can see in Figure 1).
Finally, we deleted the bottom of the mesh, that was just a big base without any interesting feature.

2 Generalization of the height based remeshing algorithm

During the exercise sessions, we implemented a remeshing algorithm that chooses the face area in function of the height. For this project, we found interesting and useful to implement a generalization of this algorithm that accepts any given vector as remeshing direction.

In other words, instead of the remeshing direction being [0, 1, 0], we can use any $[v_x,\ v_y,\ v_z]$. To do so, we compute for all vertex the new "height", described as the dot product between the normalized direction vector and the vertex position. We then retrieve the maximum and minimum between all these heights and use them to compute the normalized height for all vertices, i.e. each

one is atributed a value between 0 and 1 describing how "far" in the direction of the vector it is. For a vector [0, 1, 0] for example, the lowest vector (in the standard meaning of the word) will have the value 0 and the highest one (same) will have 1.
We then choose a minimum and maximum target length described as a ratio or a multiple of the mean target length, and map linearily the target length of the vertices to the interval between these extremums.

3 Wireframe

3.1 Idea

We wanted to modify part of the mesh to make it look like it was not yet fully rendered. The first algorithm we found was based on sphere and cylinders, but it was not satisfactory because the result didn't look sharp and didn't give the "unfinished" look we aimed for. Therefore, we tried to implement our own "wireframe" algorithm. You can see the resulting mesh on figure 2

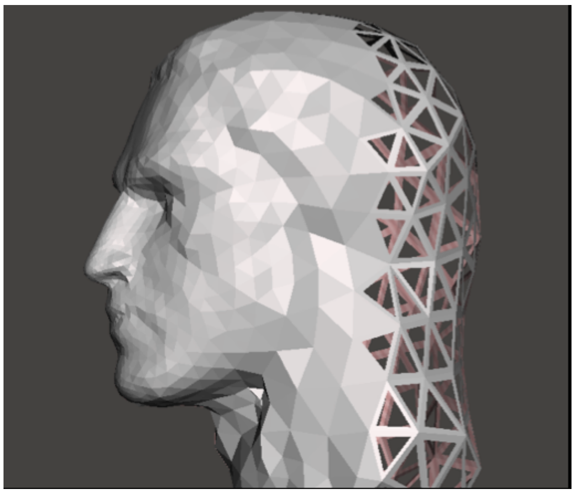

Figure 2: Remeshing front-back direction based combined with wireframe.

3.2 Algorithm

Figure 3 give an illustration for the algorithm. For any all faces, we retrieve the 3 vertices o_0, o_1 and o_2, then:

- Compute the position of center c
- For each o_i, compute the position of n_i that will form a triangle included in the first triangle
- Define m_0, m_1 and m_2 as the middle of o_0o_1, o_1o_2 and o_2o_0 respectively.
- Using the m_i positions, we build 3 faces to fulfill the final mesh.

4 Flowerize

4.1 Idea

The idea is to have other pattern than triangle as wireframe. We came up with a algorithm to transform a triangle mesh into a flower grid. We don't want flat flowers, but flowers having the curvature of the original mesh. To do this we take all the faces around a vertex and we transform these faces into a flowers shape. Two main issues occurs. First, we need to know where to construct flowers and, secondly, how to connect them. The algorithm in the next sub section takes care of this two issues.

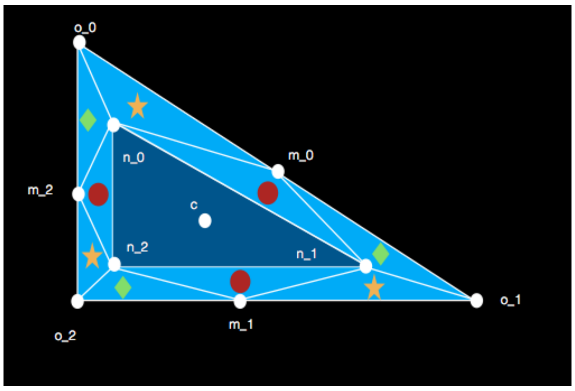

Figure 3: Wirefram algorithm

Let's discuss a little bit about some points of this algorithm. The *star* property ensure that two flowers don't intersect each others. The linearly iteration ensure that at least one petal of the flower is connect to a other flower (if the original mesh as no holes). For the connection between flowers, we use a construction that ensure that we have some thickness between two petals (see figure 4)

4.2 Algorithm

```
* For each vertex:
    * create a property, call $star$, and set it to -1
* Iterate linearly over the edges:
    * get the vertex, $vertex_{from}$, where the edge comes from
    * If the $vertex_{from}$ == -1:
        * For each neighboor vertex set $star$ property to 1
        * set $star$ property of $vertex_{from}$ to 3
* For each vertex:
    * create an hexagon (not always a hexagon)
* For each vertex with star == 3:
    * create a flowers
```

4.3 Implementation

We have some difficulties, when we create the flowers, to not create many time the same vertex . To overcomes these difficulties, we proceed in two steps. First, we create the needed vertices and, then we connect them together. We also make a heavy use of vertex property. Especially, we put vertex as properties of other vertex or edge.

5 Thickness

For the thickness, we implemented ourselves the algorithm. We first created a new vertex in the opposite direction of the vertex normal. Then we created all the faces associate to these vertices. Finally, we iterate over all boundaries and connect the two surfaces together.

We encountered one particular problem during the implementation: in very high curvature zones two neighbor vertices may have normals that go in opposite directions and it results as a conflict for edges in the output mesh. One solution we have thought (but not implemented) to overcome this problem is to smooth the vertex normal.

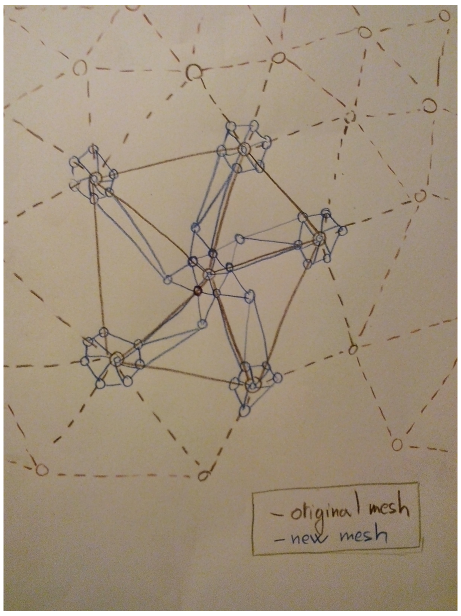

Figure 4: Flower construction

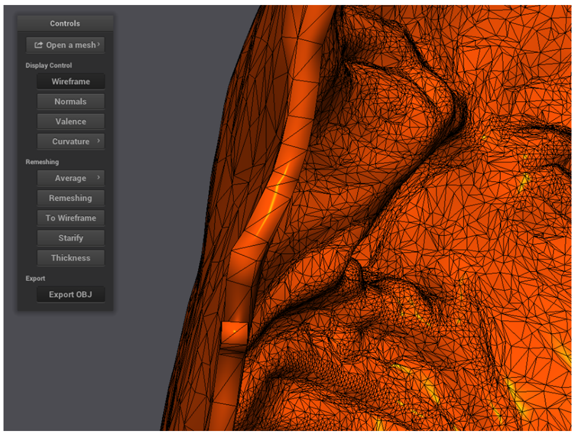

Figure 5: Thickness example on the "Max Planck" mesh

6 Support

For this part, the instruction was to create a support for a lamp that we could attach to our final mesh after the 3D printing. More specifically, we need a circular support of radius 14mm at the bottom of the Digital Geralt. We decided to create a cylinder of appropriate size that we merged with our final mesh using Meshmixer, as you can see in figure 6.

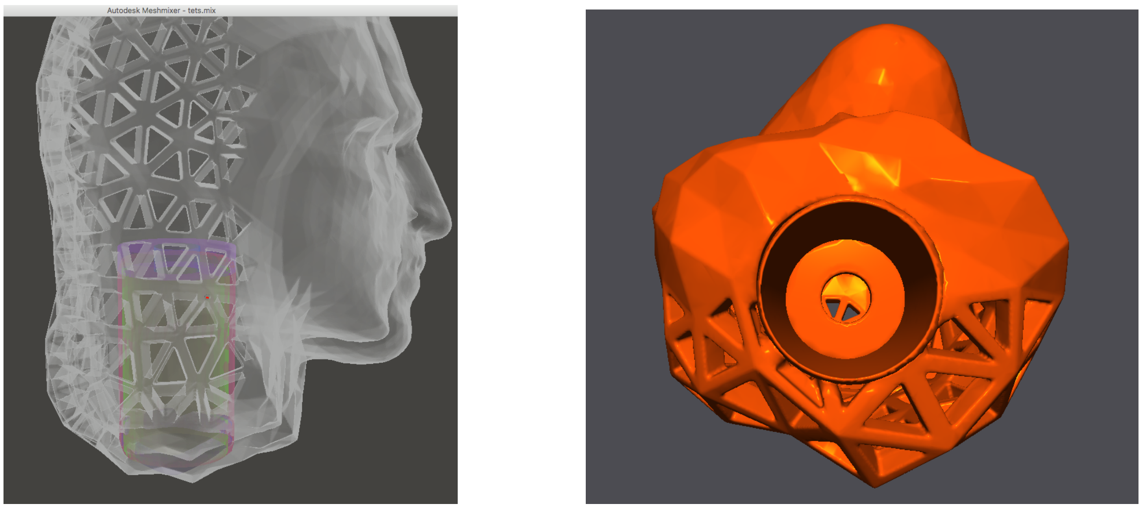

Figure 6: digital geralt with cylinder support

7 Final

For the final mesh, we decided not to use the "flowerize" algorithm decrypted in section 4 because we prefered keeping a less complex final mesh. To summarize, it combines pre-processing, directional remeshing, wireframe and thickness. The final digital geralt is shown in Figure 7.

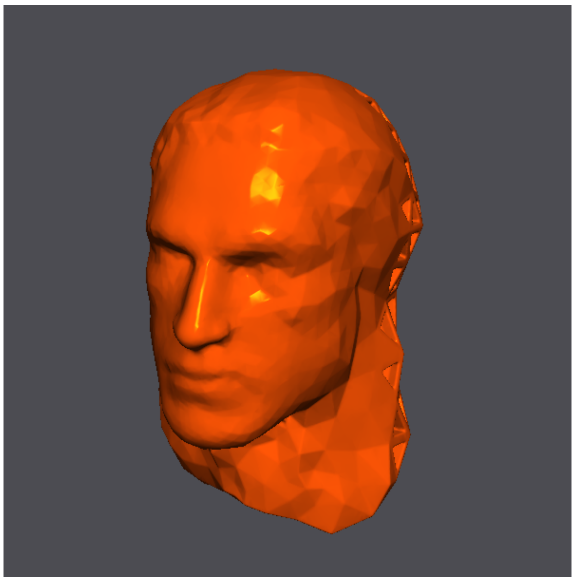

Figure 7: final digital geralt

We would like to thank the teaching team that has been very helpful and receptive during all the project, and especially our teacher Mark Pauly for all the course where we learned a lot of interesting things about meshes !