CMPT 464 Project Report

Match maker

Shuai Wu

David Stoll

Summer 2014

# Abstract

Texture mapping onto 3D meshes is used in computer graphics to enhance the visual realism of 3D models. So far there are lots of researches on how to define the texture on the 3D surface to achieve the best result. Matchmaker is a method of texture mapping which enforces positional constraints in a parameterization. It ensures to preserve as many as possible significant features of the original 3D geometry and to minimize the metrics distortion.

This project starts with an unconstrained planar embedding of 3D mesh, a 2D texture and a set of manually selected constraints. It finds out the matching triangulations between mesh and texture while satisfying the constraints.

Background

Texture mapping is a method for adding detail, surface texture, or color to a computer-generated graphic or 3D model. [2] The most common approach is to embed the mesh over a planar and to finds out the one-to-one correspondence. In the past several years, a lot of researches were done on generating a best planar parameterization with minimum distortion.

Floater proposed a method based on graph theory which aims to fit smooth surface. [3] A particular parameterization called shape-preserving is used to visually smooth surface approximations. It basically provides a parameterization by embedding the 3D mesh in the plane and using the embedding function within each mesh triangle. It requires the domain boundary to be a convex polygon, thereby increasing the distortion.

Some methods were introduced to satisfy positional constraints. One of them is to use a learning theory approach for building position correspondence. [4] It can satisfy the positional constraints in some way but it is not guaranteed to find a solution. Generating a valid mesh embedding that exactly satisfies constraints seemed to be difficult.

Introduction

Matchmaker is a new method that was introduced by Kraevoy [1], that enforces positional constraints in a parameterization. The set of constraints is manually selected on an existing planar embedding of the 3D mesh and the texture. The boundary of the planar could be any shape. With the constraints on the texture, it first triangulates the texture. Each triangle is called a patch. Then it matches each patch on the texture to a triangulation of the planar mesh formed by paths between constrained vertices. The matching triangulations are used to generate a new parameterization. The new parameterization satisfies the constraints we set at the beginning and keeps the features as many as possible.

The advantage of Matchmaker is its robustness and flexibility to a variety of 3D models and textures. Positional constraints are set and satisfied so that it can handle even the mappings requiring a large deformation. Moreover, distortion is minimized when a number of constraints are set.

Algorithms

Delaunay Triangulation

This triangulation method was used for creating the proper triangles for the constraints on the texture and for the border points that connect to the mesh.

This works by creating all possible triangles for a set of points; however, we need to make sure that the vertices of the triangles are arrange in a clockwise order. We check this by taking a random point from the three vertices and calculate the cross-product that this point forms using the two vectors that it makes to the two other points. The product is positive if the vertices are already counterclockwise otherwise it will be negative.

Now that we have all possible triangles with clockwise oriented vertices, we remove all triangles that do not satisfy a Delaunay triangulation with respect to all other points that do not make up the triangle. The method to check is from Wikipedia (<http://en.wikipedia.org/wiki/Delaunay_triangulation>):

\begin{vmatrix}
A_x & A_y & A_x^2 + A_y^2 & 1\\
B_x & B_y & B_x^2 + B_y^2 & 1\\
C_x & C_y & C_x^2 + C_y^2 & 1\\
D_x & D_y & D_x^2 + D_y^2 & 1
\end{vmatrix} = \begin{vmatrix}
A_x - D_x & A_y - D_y & (A_x^2 - D_x^2) + (A_y^2 - D_y^2) \\
B_x - D_x & B_y - D_y & (B_x^2 - D_x^2) + (B_y^2 - D_y^2) \\
C_x - D_x & C_y - D_y & (C_x^2 - D_x^2) + (C_y^2 - D_y^2)
\end{vmatrix} > 0


Basically A, B, C are the vertices in counterclockwise orientation from the triangle we check against and D is vertex in the plane. If the determinant of the matrix on the right is bigger than 0 we know that this triangle does not satisfy a Delaunay triangle since D is inside the circumstances of the points A, B and C.

Thus this triangle will be removed. In our case D, will be the vertices of all other points we used to generate triangles, except for the 3 points we used in the particular triangle. Thus, after checking all triangles for this condition, we are only left with triangles that together span the entire area without intersecting.

For creating the triangles that connect the mesh with the border points. We used a slightly smarter technique for efficiency reasons. We only created all possible triangles between the edge points and the border points, not between sets of edge points or all mesh points. Since that would generate too many cases to check.

Matching

The matching was used to find what path on the mesh we need to go to connect the constraints in the mesh in same way we do for the texture triangulation.

This was done by first pairing up the constraints on the mesh and on the texture. Then we check what edges are formed on the texture between the constraints, which tells us what constraints on the mesh need a connected path. For every triangle on the texture we find the associated constraint vertices on the mesh and try find the path between them.

This is done by an algorithm called EdgeWalker, that accepts a starting vertex and destination vertex. From the starting vertex it checks what possible edges we can walk (that are connect to this vertex) and it chooses the edge it can walk by calculating the distance between the vertex at the end of the edge and destination vertex. In this case, we always walk the shortest path to the destination.

However, there are several exceptions in which we cannot choose an edge because it fails a certain of tests. The first test, is to check if another path by another constraint pair already used the vertex of this edge that we would walk to. To avoid intersections between paths, we cannot use this edge if that is the case. In addition, we cannot go back to a vertex that we already walked on otherwise we could end up in an infinite loop that usually happen around holes in the mesh. The last check we perform is if the orientation of the vertex to the destination and all other constraint points has the same sign as the orientation between the starting constraint and destination constraint and all other constraint points. This is done by creating a vector from the vertex to the destination and a vector from the other constraint points to the vertex and calculating the cross-product. The sign from the cross-product should agree with the sign of the cross-product for the starting and destination constraint point and the associated vectors. If the signs do not agree that means that a constraint’s orientation toward the path would change in such a way that it ends up in different sub-region. This means that this edge would block this path for this constraint; thus, this vertex can’t become part of the path to avoid blocking other paths in the future.

If it passes all the tests, we can walk on the edge and keep doing this until we reach the destination or we come to a vertex where we can’t choose any valid edge because they all fail the tests. In both cases, we terminate the algorithm and return all edges and vertices that we used for the path.

Embedding

After matching, the paths between constraints are obtained and they form triangulations of the planar mesh. Each of them corresponds to a triangular patch of the texture. Now we need to find a valid embedding of the mesh that satisfies the positional constraints. It is implemented with Tutte embedding algorithm.

Basically there are two steps of embedding. The first one is to place vertices at the paths in the mesh triangulation at equal distance, along the straight line between the starting vertex and the end vertex of the path. Having the coordinates of the starting and end vertices, we can calculate the new position for each vertex along the path, and then assign the new position to them.

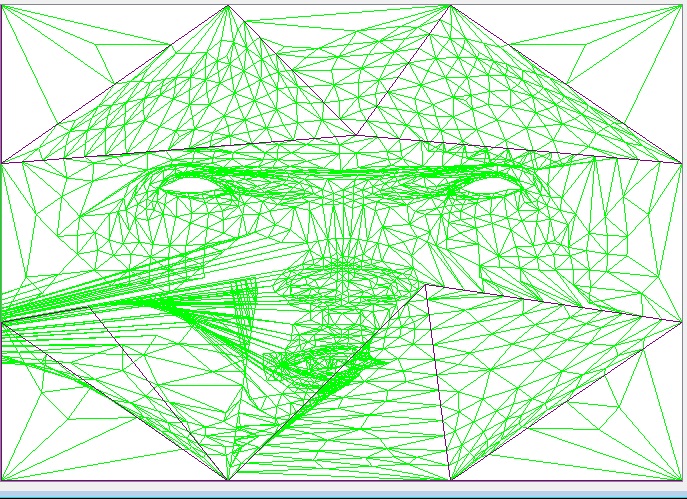
The second step is to reposition interior vertices inside triangles. For each patch, we calculate the barycentric and place all the interior vertices.

The result is a valid embedding of the mesh with vertices repositioned based on the matching triangulations.

Summary of results

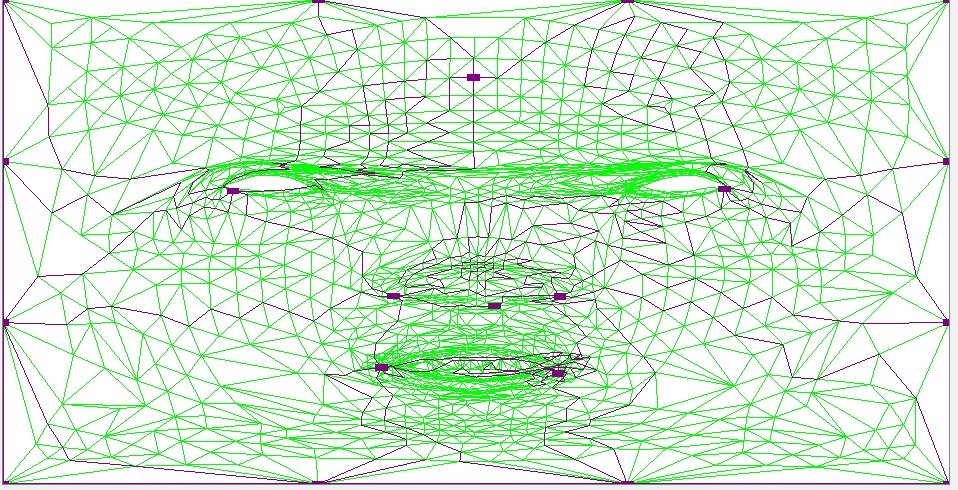
Since we could finish the matching properly we could not create a fully embedded version of the mesh, which means we could not create a texture mapping for the mesh that respects the constraints. In other words, we could observe how well the texturing would look like for this matchmaker approach.

We can speculate that that the embedding seems to work as seen in the screenshot:



Also the path finding seems to work in some cases, but others it still seems somewhat puzzling what the matching algorithm does. We suspect that either the implementation is wrong or it happens we because we did not implement the part in which we need to modify the mesh to pass the tests for the paths, which could result in rather weird paths, since algorithm might back off from the target and then approach it again as long as possible until it runs out of space and stops.

For example:



Obviously further works needs to be done to finish the matching, which must include a step to validate that the perfect case without Steiner vertices works and then the implementation should be done for the matching that insert Steiner vertices. To create path that could be not be done on the original mesh.

References

[1] Kraevoy, V., Sheffer, A., Gotsman, C. Matchmaker: Constructing constrained texture maps

[2] Wikipedia, “Texture mapping”, <http://en.wikipedia.org/wiki/Texture_mapping>

[3] FLOATER, M. S. 1997. Parameterization and Smooth Approximation of Surface Triangulation, Computer Aided Geometric Design, 14, 231-250.

[4] GUENTER, B., GRIM, C., WOOD, D., MALVAR, H., AND PIGHIN, F. 1998. Making Faces. In Proceedings of ACM SIGGRAPH 1998, Computer Graphics Proceedings, Annual Conference Proceedings, 55-66.