Realistic 3D Modeling of the Liver from MRI Images

Andrew Conegliano, Jürgen P. Schulze

University of California San Diego, La Jolla, CA, USA

Abstract. It is increasingly difficult to take care of our health with our fast paced lifestyles. If people were more aware of their health conditions, we believe change would come more easily. The Haptic Elasticity Simulator (HES) was designed so hospitals and clinics could show a patient his or her organ and poke it with a haptic device to feel its elasticity, in hopes the patient will change their lifestyle choices. This paper builds upon HES and improves the visual aspect. We discuss an end-to-end pipeline with minimal human interaction to create and render a realistic model of a patients liver. The pipeline uses a patients MRI images to create an initial mesh which is then processed to make it look realistic using ITK, VTK, and our own implementations.

1 Introduction

The Haptic Elasticity Simulator (HES) [1] is a system designed to help change the lifestyle choices of a patient who has a chronic disease. A patient can periodically monitor their health performance and see their progression of the organ with the chronic disease. The system creates a 3D model of a patients liver using his or her own Magnetic Resonance Imaging (MRI) images and displays it in 3D with a stereoscopic display. The patient can then use a haptic device to poke their organ and feel its elasticity which changes based on the organs health. When a patient can see and feel their own organ, we believe a self-realization will occur to create new healthy habits. We believe that this self-realization is much more likely to occur if the model shown to the patient looks as realistic as possible. The visual given to patients with the HES before this work was a blocky looking mesh created from segmenting a patient's MRI images. This paper improves on this mesh and proposes an end-to-end pipeline that produces a realistic looking 3D model of a liver, with minimal human interaction needed.

2 Related Work

There are many papers in the medical field relating to segmentation and model creation of the liver, specifically for the purpose of surgical training. However, these papers focus only on specific areas of the pipeline. Sorlie [2] focuses on the segmentation aspect, and for the specific purpose of tumors. His results however influenced our decision to use ITK for segmentation purposes. De Casson

et al. [3] present a system that they designed for real time deformation and realistic elasticity of the liver. It achieves this quite well, but doesn't produce a realistic model. Instead, it relies on a pre-computed liver model without a texture or shader. Finally, Neyret et al. [4] discuss a process of realistically rendering the liver. They use a Voronoi diagram for the texture as it represents a good estimate of the liver's skin texture and add specular highlights to mimic the wet surface of the liver. When applying the texture to their liver model, they use a projection approach as discussed in their earlier paper [5]. A low polygon mesh was created and projected onto a high polygon mesh using geodesics to avoid distortion, discontinuity, and repetitiveness. Their final result lacked in a key physical property of the liver however, the bumpy surface; their liver model had a plastic look. The work presented in this paper adds a normal map in the shader which accounts for the bumpy texture of the liver and produces a more realistic wet surface without looking like plastic.

3 Approach

We divide our approach to create a realistic 3D model of the liver from MRI images into four steps: Segmentation, Processing, Texture, and Lighting.

3.1 Segmentation

The first step is to take the MRI images and segment them into a 3D model. As discussed in Sorlie [2], instead of creating a new segmentation algorithm or implementing an existing one, we decided to use Kitware's open source libraries ITK and VTK. The segmentation process is from HES, but an overview is given in this paper for clarity of the end-to-end pipeline. There are four steps:

1. Stack DICOM (MRI) images into a Meta-Image (MHA) using ITK's *ImageSeriesReader* and *ImageFileWriter*.
2. Display the MRI intersections in three dimensions using a VTK widget and let the user pick three seed values for segmentation, and two values for min/max threshold intensities.
3. Once the user provides the three seed points within the organ boundaries, a lower threshold, and an upper threshold, ITK's *ThresholdSegmentationLevelSetImageFilterType* will use those values to segment connected tissue of similar intensity values and generate an MHA file.
4. VTK's *vtkContourFilter* and *vtkSTLWriter* create a mesh in StereoLithography (STL) format from the segmented images based on the contours of each image.

The resulting mesh is show in Figure 1 alongside an accurate liver model taken from BodyParts3D. The blockiness is caused by the low resolution of the MRI images. Each MRI image is $224x224$ pixels and 26 images are used to create the mesh. The overall shape of the liver however is preserved as can be seen, and with further processing a realistic mesh is created.

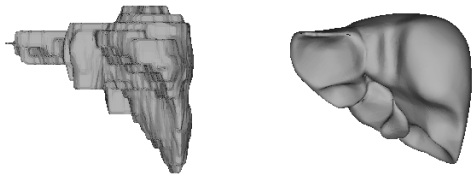


Fig. 1: Left: Segmented liver. Right: Real liver.

3.2 Processing

Once the mesh is created, it needs to be smoothed. The smoothing process is done in 8 steps:

1. Read in segmented STL mesh.
2. Convert STL mesh to VTK file format.
3. Read in VTK mesh into a quad edge mesh.
4. Smooth mesh.
5. Decimate mesh.
   (a) Create high polygon mesh with 12,000 triangles.
   (b) Create low polygon mesh with 750 triangles.
6. Write smoothed meshes to VTK file format.
7. Read in VTK smoothed meshes.
8. Convert to PLY file format.

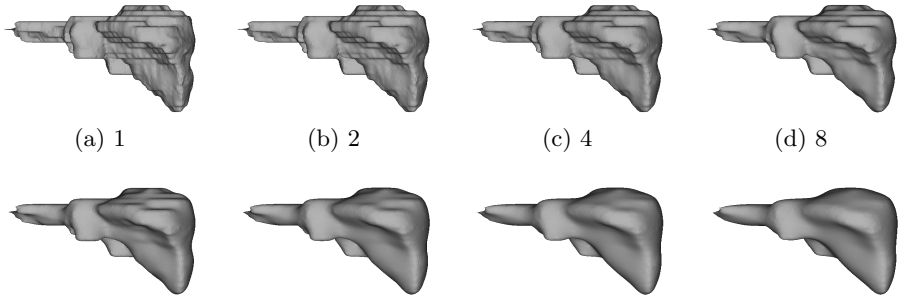
The reason for the multiple file conversion steps is because the VTK and ITK methods only work with specific input formats, and a workaround was created by converting the file multiple times. Reading the STL mesh into VTK is done using the *vtkSTLReader* class. Then the mesh is converted into a VTK mesh using *vtkPolyDataWriter*. Then we read in the VTK mesh using ITK's *MeshReaderType* into a Quad-Edge mesh object. Now we can smooth it using ITK's *QuadEdgeMeshFilter*.

ITK uses a Quad-Edge mesh for its a processing. It is a data structure that represents orientable 2-manifolds (borders of 3D solid objects) and enhances a process's speed, robustness, genericity, and maintenance cost, as described in Gouaillard et al. [6]. The reason for file format conversion is because *QuadEdgeMeshFilter* works with *Meshreader* objects, but *Meshreader* does not read in STL files. *QuadEdgeMeshFilter* then smooths the mesh using Laplacian smoothing [7]. The smoothing makes the vertices more evenly distributed and the faces better shaped. Every iteration, a vertex's position is calculated as

$$\boldsymbol{v'}_i = v_i + m_{RelaxationFactor} \cdot \frac{\sum_j w_{ij}(\boldsymbol{v_j} - \boldsymbol{v_i})}{\sum_j w_{ij}} \tag{1}$$

where w_{ij} is computed by the means of the set functor CoefficientsComputation. After each iteration, *DelaunayConformingQuadEdgeMeshFilter* is called. Delaunay triangulation means for every triangle, its circumcircle has no points in

it. An edge is considered a Delaunay edge if it is inside an empty circle. A Delaunay triangulation is conforming if every constrained edge is a Delaunay edge. The reason for this filter is because Delaunay triangulation produces better looking triangles; the triangulation forces large internal angles rather than small angles. The combination of these two algorithms produces a smoother, hence more realistic looking liver. A series of different smoothing rounds is shown in Figure 2. The more rounds the smoothing algorithm runs, the smoother the object gets. It peaks however at about 40 rounds.



(a) 1 (b) 2 (c) 4 (d) 8

(e) 16 (f) 32 (g) 50 (h) 75

Fig. 2: Mesh with x smoothing rounds

After the mesh is smoothed, it is decimated into two versions using ITK's *SquaredEdgeLengthDecimationQuadEdgeMeshFilter*. Decimation is the reduction of triangles and/or vertices in a mesh. This filter removes the shortest edge of a mesh iteratively until the targeted number of vertices is met. One version is for the final output that the patient will see, and the other is to create the texture coordinates. The reason the mesh is decimated for the final version is to get rid of any minor extremities in the mesh and produce an overall smoother looking model.

Fig. 3: Left: Squared Edge Length. Right: Quadric.

As discussed in Gelas et al. [8], there is another algorithm for decimation called *QuadEdgeMeshQuadricDecimation* which better approximates the original mesh. This removes the edge with the lowest quadric energy term. The reason

the squared edge version was chosen was to purposely lose surface information. The mesh produced by segmentation isn't very accurate as previously described. The square edge decimation smooths the mesh further, which produces a more realistic model. This can be seen in Figure 3. The left mesh appears rounder. The right mesh still has the acute edges which are caused from inaccuracy of the MRI images.

Once the mesh has finished being processed on, we need to write it to an appropriate file format so the mesh can be read later on. The Polygon File Format (PLY) was chosen because of its easy to read structure and its support for texture coordinates (the STL format is not compatible with texture coordinates). In order to write the final produced mesh, it needs to be converted to the VTK format for the same reasons as stated above when reading in the mesh. *MeshFileWriter* class is used to convert the smoothed mesh object into VTK format, *vtkPolyDataReader* is used to read in the VTK file, and finally *vtkPLYWriter* is used to convert it to PLY.

3.3 Texture

To create the texture coordinates for the mesh, the algorithm created was inspired by Neyret and Cani [5]. In their approach, they used a low polygon mesh and projected it onto a high polygon mesh using geodesics. The reason for this is to create texture coordinates that avoid too much repetition and discontinuity. If each triangle of a high polygonal mesh were to have the same texture applied, the resulting model would lose features of the texture because it is too small. To avoid this, one may use a texture small enough for each triangle so the repetition is smooth. The downside to this approach is that the texture would need to be very small depending on how many polygons there are. The approach Neyret and Cani used allows a high polygon mesh to look the same as applying a texture to a low polygon mesh, which results in less repetitions of the texture.

The approach used here is based on ray tracing concepts. Generally speaking, imagine the LPM residing inside the HPM. A ray is then shot from each vertex of the HPM and the point of intersection on the LPM is found. This point is then converted to texture coordinates for HPM. This results in the HPM having the overall same texture coordinates as the LPM. The first step is to find the closest triangles from the low polygon mesh, referred to henceforth as LPM, for each vertex in the high polygon mesh, referred to henceforth as HPM. The reason for this is to reduce the number of operations needed, which will be discussed shortly. The euclidean distance is calculated for every vertex in the LPM for each vertex in the HPM. The closest vertex found is then used to get all the triangles in the LPM that share this vertex.

Once the closest triangles in the LPM are found for a given vertex in the HPM, ray-triangle intersections are calculated. A ray is created that shoots from the HPM vertex to the LPM triangle. The ray's direction is calculated as the LPM's negative normal, and the origin is the vertex's coordinate plus an offset in the LPM positive normal direction. The offset is needed because the HPM is not guaranteed to be larger than the LPM; the offset guarantees a ray intersection.

The reason the LPM triangle's normal is used and not the normal of the HPM vertex is to avoid distortion and inaccuracy. To illustrate this, Figure 4 displays the problem with using the HPM normals as the ray direction.

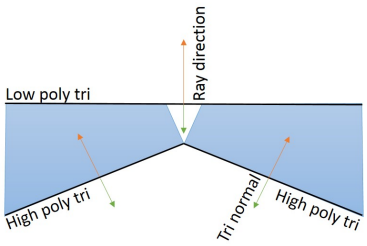


Fig. 4: Top down view of two HPM triangles and one LPM triangle.

The figure shows a top down view of two HPM triangles and one LPM triangle. The blue area represents the area that all the rays would pass through onto the LPM triangle. As can be seen, this results in a gap in the middle. Using the negative normal of the LPM solves this problem. More importantly, this also negates any distortion that might be caused from the HPM. Since the HPM mesh has many more triangles that fit into one LPM triangle, their angles and shape do not affect how the texture is applied.

Once the ray is created, the ray-triangle intersection is performed for all the triangles closest to the vertex. A point in a triangle is represented using Barycentric coordinates. To specify a point inside a triangle, the Barycentric coordinates α and β are calculated to specify point q, where

$$q = a + \alpha(b - a) + \beta(c - a) \tag{2}$$

To find an intersection, we substitute the ray equation

$$p + td \tag{3}$$

where p is the ray origin, t is the distance traveled along the ray, and d is the ray direction, into q.

$$p + td = a + \alpha(b - a) + \beta(c - a) \tag{4}$$
$$p - a = -td + \alpha(b - a) + \beta(c - a) \tag{5}$$

This results in a linear system, $X\theta = y$, where

$$X = \begin{bmatrix} -d & (b - a) & (c - a) \end{bmatrix} \tag{6}$$
$$y = [p - a]^T \tag{7}$$
$$\theta = \begin{bmatrix} t & \alpha & \beta \end{bmatrix}^T \tag{8}$$

This system of equations can be solved using Cramer's rule.

$$det(M) = -d \cdot ((b-a) \times (c-a)) \tag{9}$$
$$t = (p-a) \cdot ((b-a) \times (c-a))/det(M) \tag{10}$$
$$\alpha = -d \cdot ((p-a) \times (c-a))/det(M) \tag{11}$$
$$\beta = -d \cdot ((b-a) \times (c-a))/det(M) \tag{12}$$

In our case, we can omit the calculation of t, the distance along the ray of the intersection. We are only interested in the Barycentric coordinates. The resulting α and β is the point of intersection on the LPM. We convert this coordinate to texture coordinates with

$$u = (1-\alpha-\beta) * u_a + \alpha * u_b + \beta * u_c \tag{13}$$
$$v = (1-\alpha-\beta) * v_a + \alpha * v_b + \beta * v_c \tag{14}$$

This ray-triangle intersection routine is based on the Möller-Trumbore algorithm [9]. The vertex of the HPM is finally assigned the texture coordinates (u, v) as calculated above. The (u, v) coordinates are used for all triangles in the HPM that fully overlap a triangle in the LPM - three rays shot from the HPM triangle all intersect the same LPM triangle. For the case when a HPM triangle does not fully overlap into a LPM triangle, the texture coordinates are calculated by an approximation. When one or two vertices of a HPM triangle do not overlap the same LPM triangle, their Barycentric coordinates do not satisfy equations (2), (3), and (4), which means that the point is outside the triangle. The negative coordinates however are used. If we were to recalculate independently for every vertex that does not intersect the same LPM triangle, the resulting texture coordinates would be severely distorted.

Each vertex is independently calculated, which happens to result in the same Barycentric coordinates for each vertex. This results in the same texture coordinates for each vertex, which displays a solid color for that triangle. The reason we keep the negative texture coordinates is because the OpenGL *repeat* attribute for textures is taken advantage of.

3.4 Lighting

The final step in our pipeline for a realistic liver is the lighting. The normals of the mesh are smoothed and a normal map and a BRDF was implemented in GLSL to create a wet and bumpy appearance for the model. To smooth the normals of the mesh, OpenSceneGraph's *SmoothingVisitor* class was used. The class computes the normal for each vertex by averaging the facet normals of shared polygons. When light is reflected on the triangles, the intensity values between triangles have a smaller delta, which results in smoother looking shadows. A normal for each texture was created using the Linux tool GIMP. Lastly, the BRDF implemented is the Cook-Torrance model [10], which focuses on the specular component by treating the surface as having many microfacets. The estimated Cook-Torrance equation is as follows

$$\boldsymbol{r} = ambient + \sum_{l} \boldsymbol{n} \cdot \boldsymbol{l}(k + (1-k)r_s) \tag{15}$$

where k is the fraction of diffused light and r_s is the specular component. The specular component is computed as

$$r_s = \frac{FDG}{\pi(n \cdot l)(n \cdot v)} \tag{16}$$

where F is the Fresnel factor, D is the directional distribution of the microfacets known as the roughness, and G is the geometric attenuation. The Fresnel factor describes the behavior of light moving between different refractive mediums. It defines the fraction of incoming light that is transmitted and reflected. The Fresnel equation is computational expensive so Schlick's approximation [11] is used

$$F_\lambda(u) = f_\lambda + (1 - f_\lambda)(1 - (h \cdot v))^5 \tag{17}$$

where f_λ is the reflectance at normal incidence, l is the light vector, and h is the half-vector between the light vector and the vector pointing towards the viewer. The roughness factor describes the distribution of microfacets that are pointed in the same direction, and hence their normals are the same, around a given direction. For a rough surface, the distribution is higher which leads to larger variances in microfacet normals, and for a smooth surface, the distribution is low which leads to a low variance of microfacet normals. The roughness factor is calculated using the Beckmann distribution function [12]

$$D = \frac{1}{\pi m^2 \cos^4 \alpha} e^{-(\frac{\tan \alpha}{m})^2} = \frac{1}{\pi m^2 (\boldsymbol{n} \cdot \boldsymbol{h})^4} e^{(\frac{(\boldsymbol{n} \cdot \boldsymbol{h})^2 - 1}{m^2 (\boldsymbol{n} \cdot \boldsymbol{h})^2})} \tag{18}$$

where m is a variable that defines how rough the surface is.

The last part of the BRDF is the geometric attenuation, calculated using Blinn's model [13]. This describes the proportion of light that remains after some of the microfacets have blocked light reaching the surface or the light reflected, assuming each microfacet is a V shaped groove. By describing the microfacets this way, three different cases of how light reacts with the surface can happen:

a) The light is reflected with no obstructions.
b) Some of the incoming light is blocked before reaching the surface.
c) Some of the reflected light is blocked after reflection.

Each case is described by the following calculations

$$G_a = 1 \tag{19}$$

$$G_b = \frac{2(\boldsymbol{n} \cdot \boldsymbol{h})(\boldsymbol{n} \cdot \boldsymbol{v})}{\boldsymbol{v} \cdot \boldsymbol{h}} \tag{20}$$

$$G_c = \frac{2(\boldsymbol{n} \cdot \boldsymbol{h})(\boldsymbol{n} \cdot \boldsymbol{l})}{\boldsymbol{l} \cdot \boldsymbol{h}} \tag{21}$$

$$G = min(G_a, G_b, G_c) \tag{22}$$

4 Implementation Details

The project was written in Visual C++ using Microsoft Visual Studio 12 x64, ITK 4.7.1, VTK 6.2.0 and OpenSceneGraph (OSG) 3.2.1. A custom converter was built to convert from PLY to OSG's native format OSGT. Eight point lights were added to the scene, in a cube orientation, around the model to produce many specular highlights and increase the brightness of the texture. The final parameters used to produce the model are shown in Table 1.

Smoothing rounds	25
Low Polygon Triangle count	750
High Polygon Triangle count	12,000
f_λ	.09
m (roughness)	.03
k (fraction of diffused light)	.25

Table 1: Final parameters used.

5 Results

(a) (b) (c)

Fig. 5: Final model output, with Visible Korean Project texture.

The result is shown in Figure 5. To show how the reflection looks, the model is slightly rotated for every picture. To create the wet look, the specular component was maximized by minimizing the roughness and Fresnel parameters. The texture used for the final model is from a picture of a slice of the human body created by The Visible Korean Project [14]. The model can be rotated in real time and easily achieves a stable 30 FPS. Total run time of all steps takes under 5 minutes. Program was run and compiled using a Core-i7-4702HQ @ 2.20GHz with 8GB RAM on Windows 8.1 x64.

6 Conclusion

Living a healthy lifestyle is becoming a difficult task in today's society. We hope that if a patient is able to see his or her own liver, the chance for a change in their lifestyle will dramatically increase, with the goal of preventing or controlling chronic liver diseases. By creating an end-to-end pipeline built into HES, where minimal human interaction is needed, we offer something that is unique and viable to be used by hospitals or clinics. A realistic model of the liver is successfully created by segmenting the MRI images, smoothing the mesh, adding a texture, and finally creating a shader that applies specular highlights to simulate a wet surface. Our approach can theoretically be applied to other organs as well. More research is needed, however, to confirm this.

References

1. Alghamdi, A.A.: Hes: Haptic elasticity simulator. Master's thesis, University of California, San Diego (2014)
2. Sorlie, R.P.: Automatic segmentation of liver tumors from mri images. Master's thesis, University of Oslo (2005)
3. de Casson, F.B., dAulignac, D., Laugier, C.: An interactive model of the human liver. In: Experimental Robotics VII. Springer (2001) 427–436
4. Neyret, F., Heiss, R., Sénégas, F.: Realistic rendering of an organ surface in real-time for laparoscopic surgery simulation. The Visual Computer **18** (2002) 135–149
5. Neyret, F., Cani, M.P.: Pattern-based texturing revisited. In: SIGGRAPH 99 Conference Proceedings, ACM SIGGRAPH, Addison Wesley (1999) 235–242
6. Gouaillard, A., Florez-Valencia, L., Boix, E.: Itkquadedgemesh: A discrete orientable 2-manifold data structure for image processing. (2006)
7. Sorkine, O., Cohen-Or, D., Lipman, Y., Alexa, M., Rössl, C., Seidel, H.P.: Laplacian surface editing. In: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing, ACM (2004) 175–184
8. Gelas, A., Gouaillard, A., Megason, S.: Surface meshes incremental decimation framework. Insight J (2008) 1–8
9. Möller, T., Trumbore, B.: Fast, minimum storage ray/triangle intersection. In: ACM SIGGRAPH 2005 Courses, ACM (2005) 7
10. Cook, R.L., Torrance, K.E.: A reflectance model for computer graphics. In: ACM Siggraph Computer Graphics. Volume 15., ACM (1981) 307–316
11. Schlick, C.: An inexpensive brdf model for physically-based rendering. In: Computer graphics forum. Volume 13., Wiley Online Library (1994) 233–246
12. Beckmann, P., Spizzichino, A.: The scattering of electromagnetic waves from rough surfaces. Norwood, MA, Artech House, Inc., 1987, 511 p. **1** (1987)
13. Blinn, J.F.: Models of light reflection for computer synthesized pictures. In: ACM SIGGRAPH Computer Graphics. Volume 11., ACM (1977) 192–198
14. Park, J., Chung, M., Hwang, S., Lee, Y., Har, D., Park, H.: Visible korean human - improved serially sectioned images of the entire body. IEEE Trans Med Imaging 24 (2005) 352–360