FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Splitting of Meshes in Image-Space

BACHELORARBEIT

zur Erlangung des akademischen Grades

Bachelor of Science

im Rahmen des Studiums

Medieninformatik und Visual Computing

eingereicht von

Barbara Schwankl
Matrikelnummer 0852176

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Mitwirkung: Projektass.(FWF) Ing. Peter Mindek, MSc

Wien, 05.04.2013

(Unterschrift Verfasserin) (Unterschrift Betreuer)

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Splitting of Meshes in Image-Space

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Media Informatics and Visual Computing

by

Barbara Schwankl
Registration Number 0852176

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Eduard Gröller
Assistance: Projektass.(FWF) Ing. Peter Mindek, MSc

Vienna, 05.04.2013

(Signature of Author) (Signature of Advisor)

Erklärung zur Verfassung der Arbeit

Barbara Schwankl
Unionstr. 133, 4020 Linz

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum)	(Unterschrift Verfasserin)

Inhaltsverzeichnis

KAPITEL 1

Introduction

The main purpose of 3D computergraphics is to represent image data. To make these data examinable, user interaction becomes an important part. Hence, the user should be able to translate, rotate and scale the 3D object. With the complexity of the object, the desire to examine just parts of it rises. The examination of the objects' inner structures can also be of interest. That leads to the first problem that needs to be solved: How can we reveal the inner sturctures? By simply removing the outer structures, the context would also get lost that could be important for scientific findings. Hence, the idea is to reveal the point of interest whilst keeping the context. Therefor, several scientific approaches exist. One idea is to simply raise the opacity of the outer structures. For example to reveal a brain, the approach would be to simply raise the opacity of the scull and the skin respectively for making the parts of secondary interest semi-transparent. Another approach is to cut the object so that no information would get lost and still the inside would be fully visible. So called explosion views unfold the inside by breaking the outer struktures into multiple parts and shifting those apart whilst keeping the point of interest in the center.

This paper will examine several approaches to reveal inner structures of complex 3D models retaining the context to be able to examine the object regarding its surroundings.
TOEDIT In section BLA you will learn how to BLA and BLA BLA

KAPITEL 2

Analysis of existing approaches

2.1 Literature studies

2.2 Analysis

2.3 Comparison and summary of existing approaches

KAPITEL 3

Methodology

3.1 Plug-ins in VolumeShop

VolumeShop is an interactive hardware-accelerated application for direct volume illustration (Bruckner, Gröller [1]). It is designed for developers to have maximum flexibility in use for vizualisation research. Its objects can be dynamically created and accessed (http://www.bg.tuwien.ac.at/volumeshop, 101).

Plug-ins are functionally independent and can be dynamically loaded. One main advantage in development is that the application does not need to be closed when the plug-in is recompiled.

Properties

The complete state of a plug-in is defined by its properties which constitute the plug-ins' functionality.
A plug-in can be easily created. An example for an integer property in the range [0,255]:

```
GetPlugin().GetProperty("‘Test2"’) = Variant::TypeInteger
   (12,0,255);
```

For extended functionality there is also the possibility of linking properties. The change of a property causes linked properties to change as well.
Creating links in the GUI: [FIGURE] Creating links programmatically:

```
// Link property "‘MyProperty"’ to property "‘LinkedProperty"’
PropertyContainer::Link myLink(pTargetObject,"’LinkedProperty
   "’);
GetPlugin().SetPropertyLink("‘MyProperty"’,myLink);
```

Observers

Observers allow tracking changes in properties or other objects. Notifications are being bound to member functions with the class 'ModifiedObserver'. This class notifies changes from multiple objects of different types.
An example for using observers:

```
// usually a class member
ModifiedObsever myObserver;

// typically in plugin constructur
// connect observer to member function
myObserver.connect(this,&MyPlugin::changed);

// add observer to objects we want to track
GetPlugin().GetProperty("‘MyProperty1"’).addObsever(&myObserver
   );
GetPlugin().GetProperty("‘MyProperty2"’).addObsever(&myObserver
   );

// notification handler
void changed(const Variant & object, const Observable::Event &
   event)
{
      // handle changes, e.g., trigger re-render
      GetPlugin().update();
}
```

For this work the display() function is of most importance. It is responsible for the rendering.

Languages

The programming of the plug-ins is done in C++ using OpenGL that is a successful cross-platform graphics application programming interface (API) for 2D and 3D computer graphics [BOOK hill p preface v].
For shading and texturing OpenGL Shading Language (GLSL) is used.

3.2 Concept of Shaders

In GLSL there are two main shaders:

- Vertex shader
- Fragment shader

In 3D computer graphics objects are described with a set of polygon surface patches and are called 'polygonal mesh' or simply 'mesh'. Each polygon has several vertices, edges and faces [BOOK hearn baker p 123, 124]. With GLSL the shading of the polygons can be modified directly, replacing the default shading function of OpenGL.

Vertex shader

The vertex data is taken as input. These data include position, color and normals.
The vertex shader performs tasks such as:

- transforming vertex positions
- transforming the normal vectors and normalizing them
- generating and transforming texture coordinates
- applying light models such as ambient, diffuse and specular per vertex
- computing color

[BOOK hill p 439]

Fragment shader

After the vertices have been transformed into the viewplane they are rasterized. The result are fragments which contain information about screen coordinates, depth, color, texture coordinates and so on. The value of the pixel color is determined by interpolation of the vertex colors.
The fragment shader performs tasks such as:

- applying light values
- computing shadows
- adding complex texture(e.g. Bump Mapping)

[BOOK hill p 440]

KAPITEL 4

Suggested implementation

In this section a step by step approach to reveal inner structures of a mesh will be discussed. First a plane needs to be defined that represents the position and direction of the cut. To retain interactivity, several parameters in the VolumeShop interface are available to translate, rotate and scale the plane. The color and opacity of the plane should be adaptable to not occlude parts of the mesh. For the mesh splitting an offset is defined that indicates how far the two halfs of the mesh shall be apart from the plane. The larger the gap the better the insight into the model. The splitting itself is no real translation of the two halfs, but the model is rendered twice at different positions in space parallel to the plane. The final step is shading of the models' surface as well as the back facing triangles. The latter will be shaded with a different color, but with the same amount of lighting to create the illusion of depth.

4.1 Definition of the plane

A plane in this context is a square of infinite size. It is used as a visual help to define where the mesh will be cut.
To be able to intervene with VolumeShop, it is necessary to provide properties to set the translation, rotation, and scale vectors as well as the color. Each property requires a name and a type. Here an example of a property named „Plane Translation Vector" of the type Vector (in this case vec3):

```
GetPlugin().GetProperty("Plane Translation Vector").require(
   Variant::TypeVector(Vector(0.0f, 0.0f, 0.0f)));
```

To apply changes made in the interface immediately, an observer for the property has to be added:

```
GetPlugin().GetProperty("Plane Translation Vector").addObserver
   (&m_modVariantObserver);
```

Before drawing the plane, the Viewing Transformation Matrix is loaded. This matrix is for tranforming the coordinates from world space into viewing space.[BOOK: Hearn, Baker]
Then the plane is being adjusted by its affine transformations. All passed parameters are of the type 'float' and taken from the user input.
The commands in OpenGL:

```
glTranslatef(vecPlaneTranslation.GetX(), vecPlaneTranslation.
   GetY(), vecPlaneTranslation.GetZ());
glRotatef(vecPlaneRotationAngle, vecPlaneRotation.GetX(),
   vecPlaneRotation.GetY(), vecPlaneRotation.GetZ());
glScalef(vecPlaneScaling.GetX(), vecPlaneScaling.GetY(),
   vecPlaneScaling.GetZ());
```

The color of the plane is handed over from the input panel, normalized, and passed on to the renderer. In VolumeShop, this can be easily archieved by calling the function 'GetNormalized<ColorChannel>()'.

```
glColor4f(vecPlaneColor.GetNormalizedRed(), vecPlaneColor.
   GetNormalizedGreen(), vecPlaneColor.GetNormalizedBlue(),
   vecPlaneColor.GetNormalizedAlpha());
```

OpenGL supports several basic graphics primitives by default. For the plane a quad is required that looks like the following [BOOK hill, p70]:

```
glBegin(GL_QUADS);
      glNormal3f(0, 0, 1);
      glVertex3f(-1, -1, 0);
      glVertex3f( 1, -1, 0);
      glVertex3f( 1, 1, 0);
      glVertex3f(-1, 1, 0);
glEnd();
```

As stated above our plane will be indefinite, but for the purpose as a visual helper it is displayed from -1 to 1 as a default. Note that the plane is also scaleable for bigger meshes.

The result should look like [FIGURE].

4.2 Splitting the mesh

The splitting offset is again implemented as a interface parameter. If the parameter is set to '0.0f' the mesh does not seem to be split. The larger the value of the offset the bigger the distance between the two halfs of the mesh.

```
GetPlugin().GetProperty("Offset").require(Variant::TypeFloat
   (0.5f));
```

Note that of course an observer needs to be added as well. (REF 4.1.)

Rotation of the planes' normal vector

The normal of the plane has been defined with '(0.0, 0.0, 1.0)', but regarding that the normal vector is still located in the object space of the plane, it needs to be transformed into the same space as the mesh. This is being done by rotating the normal by the same amount as the plane is rotated in the interface.
The following code reveals the rotation matrix regarding the interface inputs.

```
glLoadIdentity();
glRotatef(vecPlaneRotationAngle, vecPlaneRotationVector.GetX(),
    vecPlaneRotationVector.GetY(), vecPlaneRotationVector.GetZ
   ());
```

Of course, the rotation matix could also be calculated using the generally known formulas[BOOK hill p 217]:

x-roll:

$$R_x(\beta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\beta) & -\sin(\beta) & 0 \\ 0 & \sin(\beta) & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.1}$$

y-roll:

$$R_y(\beta) = \begin{pmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.2}$$

z-roll:

$$R_z(\beta) = \begin{pmatrix} \cos(\beta) & -\sin(\beta) & 0 & 0 \\ \sin(\beta) & \cos(\beta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.3}$$

However, it should be considered that 3D rotation matrices do not commute [BOOK hill p220], so the order of multiplication matters. For details on matrix multiplication and applying affine transformations please look up [BOOKS]
Next the rotated normal vector should be multiplied with the modelview matrix to get the vector in modelview space before the it is normalized. After normalization the normal vector has a value between 0 and 1.

```
planeNormal.normalize();
```

Translating the mesh

After the light calculations the mesh is translated by the value of the offset in the direction of the normal vector. The command in OpenGL for translation is 'glTranslatef'. Before the mesh is rendered the shader needs to be bound.

```
Vector meshTranslation = planeNormal * offset;
glTranslatef(meshTranslation.GetX(), meshTranslation.GetY(),
   meshTranslation.GetZ());
m_shaShader.bind();
renderMesh(*pMesh);
m_shaShader.release();
```

As stated above the mesh is rendered twice to create the illusion of a mesh that is dragged apart. So the mesh needs to be rendered again with a translation in the opposite direction.
In VolumeShop, the two meshes displayed with a plane in between. The distance of the meshes is twice the offset [FIGURE].

Discarding dispensable pixel

To eliminte the pixels from the other side of the plane of each mesh respectively, it has to be determined on which side of the plane a pixel lies.
The computation is performed in the fragment shader.
According to [BOOK], the dot product of the plane normal and the vertex position minus a point on the plane states if the vertex is in front of or behind the plane. If the resulting value is smaller or equal to zero, the point is behind the plane and should not be drawn. Before the mesh is rendered, the normal of the plane as well as a point on the plane must be passed to the shader as uniforms. For the first half of the mesh the normal vector is untouched, for the second one converted. The point on the plane has to be determined in respect of the plane translation.

Definition of the plane normal and point on the plane as uniforms:

```
glUniform3f(m_shaShader.GetUniformLocation("planeNormal"),
   planeNormal.GetX(), planeNormal.GetY(), planeNormal.GetZ());
glUniform3f(m_shaShader.GetUniformLocation("planePoint"),
   vecPlaneTranslation.GetX(), vecPlaneTranslation.GetY(),
   vecPlaneTranslation.GetZ());
```

The fragment shader discards the points behind the plane:

```
float dotProduct = dot((vertexPosition - pointOnPlane),
   planeNormal);

if(dotProduct <= 0.0) {
      discard;
}
```

4.3 Shading the mesh

The back faces of the mesh still have the same color as the front faces, what makes it look unreal. Therefor the back faces have to be shaded in a different color. OpenGL has a function to check if the fragment is front facing:

```
if(!gl_FrontFacing){
      gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0) //shades all
         inner facing vertices red
}
```

Using the phong shading model for the back facing fragments

To add the impression of depth, lighting of the back facing fragments is being done with the phong shading model [BOOK].

```
void directionalLight(in gl_LightSourceParameters light, in vec
   3 N, in vec3 V, in float shininess, inout vec4 ambient,
   inout vec4 diffuse, inout vec4 specular)
{
      vec3 L = normalize(light.position.xyz);

      float nDotL = dot(N, L);

      if (nDotL > 0.0)
      {
            vec3 H = normalize(light.halfVector.xyz);

            float pf = pow(max(dot(N,H), 0.0), shininess);

            diffuse += light.diffuse * nDotL;
            specular += light.specular * pf;
      }

      ambient += light.ambient;
}

void main()
{
      vec3 v = normalize(vVertexPosition);
      vec3 n = normalize(vVertexNormal);

      if(!gl_FrontFacing) {
```

```
            directionalLight(gl_LightSource[0], n, v, gl_
               FrontMaterial.shininess, ambient, diffuse,
               specular);
            color.rgb = ((ambient * vec4(1.0, 0.1, 0.0, 1.0)) +
                (diffuse * vec4(1.0, 0.1, 0.0, 1.0)) + (
               specular * vec4(1.0, 1.0, 1.0, 1.0))).rgb;
            color.a = 1.0;
            color = clamp(color, 0.0, 1.0);
      }
      gl_FragColor = color;
}
```

KAPITEL 5

Critical reflection

KAPITEL 6

Summary

KAPITEL 7

Bibliography

7.1 Literature Search

Information on online libraries and literature search, e.g., interesting magazines, journals, conferences, and organizations may be found at `http://www.big.tuwien.ac.at/teaching/info.html`.

7.2 BibTeX

BibTeX should be used for referencing.

The LaTeXsource document of this pdf document provides you with different samples for references to journals [3], conference papers [6], books [2], book chapters [7], electronic standards [5], dissertations [8], masters' theses [4], and web sites [1]. The respective BibTeX entries may be found in the file `references.bib`. For administration of the BibTeX references we recommend `http://www.citeulike.org` or JabRef for offline administration, respectively.

Literaturverzeichnis

[1] Business Informatics Group. http://www.big.tuwien.ac.at. Accessed: 2010-11-09.

[2] M. Hitz, G. Kappel, E. Kapsammer, and W. Retschitzegger. *UML @ Work, Objektorientierte Modellierung mit UML 2*. dpunkt.verlag, 3. edition, 2005 (in German).

[3] Christian Huemer, Philipp Liegl, Rainer Schuster, and Marco Zapletal. B2B Services: Worksheet-Driven Development of Modeling Artifacts and Code. *Computer Journal*, 52(2):28–67, 2009.

[4] P. Langer. Konflikterkennung in der Modellversionierung. Master's thesis, Vienna University of Technology, 2009.

[5] OASIS. *Business Process Execution Language 2.0 (WS-BPEL 2.0)*, 2007.

[6] A. Schauerhuber, M. Wimmer, W. Schwinger, E. Kapsammer, and W. Retschitzegger. Aspect-Oriented Modeling of Ubiquitous Web Applications: The aspectWebML Approach. In *Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS '07), March 26-29, Tucson, Arizona, USA*, pages 569–576. IEEE CS Press, 2007.

[7] W. Schwinger and N. Koch. Modeling Web Applications. In G. Kappel, B. Pröll, S. Reich, and W. Retschitzegger, editors, *Web Engineering*, pages 39–64. John Wiley & Sons, Ltd, 2006.

[8] M. Wimmer. *From Mining to Mapping and Roundtrip Transformations - A Systematic Approach to Model-based Tool Integration*. PhD thesis, Vienna University of Technology, 2008.