

# Visualization of Composite Objects Through Techniques of Exploded Views and Ghosting

BACHELORARBEIT

zur Erlangung des akademischen Grades

**Bachelor of Science**

im Rahmen des Studiums

**Medieninformatik und Visual Computing**

eingereicht von

**Hartwig Wutscher**

Matrikelnummer 0426961

an der Fakultät für Informatik  
der Technischen Universität Wien

Betreuung: o.Univ.-Prof. Dipl.-Ing. Mag. Dr. **Monika Musterprofessorin**

Mitwirkung: **Ma. Sc.** Peter Mindek  
Supervision assistant 2

Wien, 3.9.2014

\_\_\_\_\_  
(Unterschrift Verfasser)

\_\_\_\_\_  
(Unterschrift Betreuer)



# Visualization of Composite Objects Through Techniques of Exploded Views and Ghosting

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

**Bachelor of Science**

in

**Media Informatics and Visual Computing**

by

**Hartwig Wutscher**

Registration Number 0426961

to the Faculty of Informatics  
at the Vienna University of Technology

Advisor: o.Univ.-Prof. Dipl.-Ing. Mag. Dr. Monika Musterprofessorin

Assistance: Ma. Sc. Peter Mindek

Supervision assistant 2

Vienna, 3.9.2014

\_\_\_\_\_  
(Signature of Author)

\_\_\_\_\_  
(Signature of Advisor)



# Erklärung zur Verfassung der Arbeit

Hartwig Wutscher  
Nesselgasse 4/22, 1170 Wien

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 Verfasser)



# Kurzfassung

Die Explosionsgrafik ist eine Technik der Illustrativen Visualisierung, die die Funktion oder den Aufbau eines komplexen Objekts darstellt, in dem es in seine Einzelteile zerlegt wird, die dann so im Raum platziert werden, dass sie im Idealfall vollständig sichtbar sind und sich durch ihre Position auf ihre ursprüngliche Position innerhalb des Objekts rückschließen lässt. Diese Bachelorarbeit beschäftigt sich mit der Implementierung eines Plug-Ins für die Visualisierungssoftware VolumeShop, das in der Lage ist aus zusammengesetzten Dreiecks-Meshes einfache Explosionsgrafiken dynamisch zu erzeugen. Um aussageschwache Resultate bei Betrachtung von bestimmten Blickwinkeln aus zu vermeiden habe ich für diese Fälle Explosionszeichnung mit Ghosting kombiniert. Ghosting ist eine andere Illustrationstechnik, bei der Objekte, die sich zwischen dem Betrachter und einem wichtigen Objekt befinden transparent dargestellt werden.





# Abstract

Explosive views are a technique in illustrative visualization where the inner working of complex objects is revealed by segmenting it into several parts that are then displaced so that ideally they are fully visible and their positions convey information about their original place in the object. This thesis deals with implementing a plug-in for the visualization software Volume Shop that is able to create simple dynamic exploding views from composite triangle meshes. To avoid behavior that renders the graphic useless when viewed from certain viewpoints, I combined exploded views with ghosting, which is a different illustrative technique where an object between the viewer and a more important object is drawn translucently.

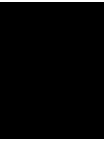




# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>State of the Art Description</b>	<b>3</b>
2.1	What are the main problems when creating exploded views? . . . . .	3
2.2	Analysis . . . . .	7
<b>3</b>	<b>Practical part</b>	<b>9</b>
3.1	Plan and milestone definition: . . . . .	9
3.2	Tools and languages . . . . .	9
3.3	Documentation of the implementation each milestone . . . . .	9
<b>4</b>	<b>Conclusion</b>	<b>15</b>
	<b>Bibliography</b>	<b>17</b>





# Introduction

The aim of illustration is to generate expressive images that effectively convey certain information via the visual channel to the human observer.



Illustrations **have been** since the paleolithic age [7] to give their viewers a graphical representation of things seen, experienced or imagined [9]. Since the development of perspective and the necessity to intuitively explain complex technical, medical and scientific matters, especially since the industrial revolution, several techniques to successfully achieve this have been developed.

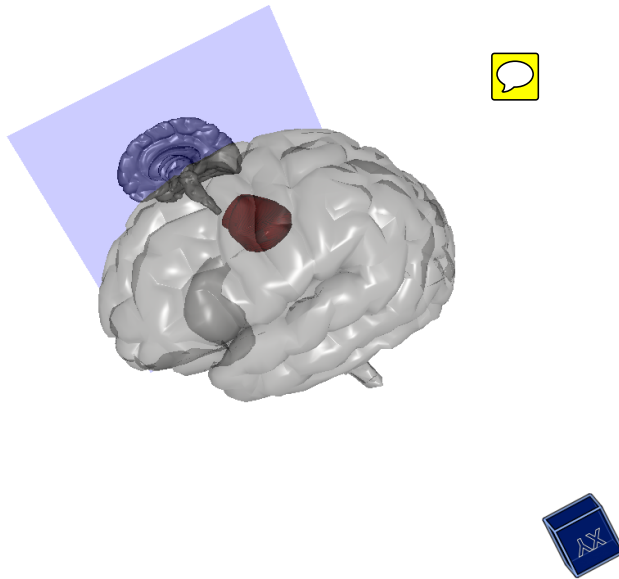
Illustrative techniques are often designed in a way that even a person with no technical understanding clearly understands the piece of art. [7]


With the arrival of computers that are able to create complex graphics that react in real-time to user interactions, the transfer of these static illustrations to a dynamic visual representation has posed new challenges to the art of illustration. Exploded views are a technique used in illustrative visualization, where complex real world objects are drawn in a way that conveys how these objects are built, how they are assembled or how they work. Typically these objects are systems of interacting parts that may occlude each other or even be completely hidden by an outer hull of the visualized objects.

Examples for this would be a motor that consists of many independent parts, where the Illustration should give an idea of how it might work, a chest of drawers bought as a set of boards and connectors, that has a visual assembly instruction or the depiction of an anatomic feature.

In an exploded view the object is decomposed into several parts which are displaced so that internal details are visible [2]

So for technical illustrations the parts that constitute a machine could be moved on an axis so that they are all visible and their position relative to this axis would still give information as



 **Figure 1.1:** Illustration of a brain, combining ghosting and an exploded view

to where the place of each part would be inside the machine. As for the assembly instruction the displacement would happen along the axes they will be put together to make it clear to the viewer how to assemble the object.

The goal of my work was to create an interactive visualization that would dynamically generate simple exploded views. Building upon an existing plugin for the **Volumeshop** visualization platform, that splits objects along a plane and displaces the two parts, I created a plugin where the **User** can define an object of interest, that **isn't** cut and stays in place, while the split parts are displaced, revealing said object.

In an interactive System this displacement **-the** "explosion" can be shown as an animation, with all parts starting at the original place and gradually moving into the desired place. The User can also choose, which part of the system is interesting to him, and view the object from different angles, which also results in an animated change of the part's location. This can lead to massive displacement of the parts, especially if the viewer is looking in the direction of the displacement axis.


To prevent this, I designed a system with a maximum distance of displacement. To prevent complete occlusion I combined the exploded view with ghosting, which is a technique where Objects occluding other objects are being drawn transparently.

## State of the Art Description



### 2.1 What are the main problems when creating exploded views?

To create an exploded view, a few considerations have to be made Li et al. provide a comprehensive list of these challenges in their 2008 paper [3].

Most important of all is the question what Information should be transported with the image, what the purpose of the illustration is. In essence what are the objects of interest that should attract the viewers attention? This usually is determined by preliminary definition of interesting features, definitions of parts, subsets of parts of the data or user interaction while viewing the graphic. 

#### Explosion direction

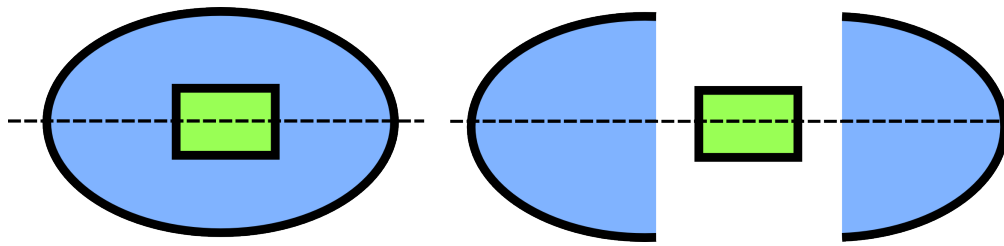
Also one has to consider in which direction the parts should be displaced.

Many objects have a canonical coordinate frame that may be defined by a number of factors, including symmetry, real-world orientation, and domain-specific conventions. In most exploded views, parts are exploded only along these canonical axes. [3]

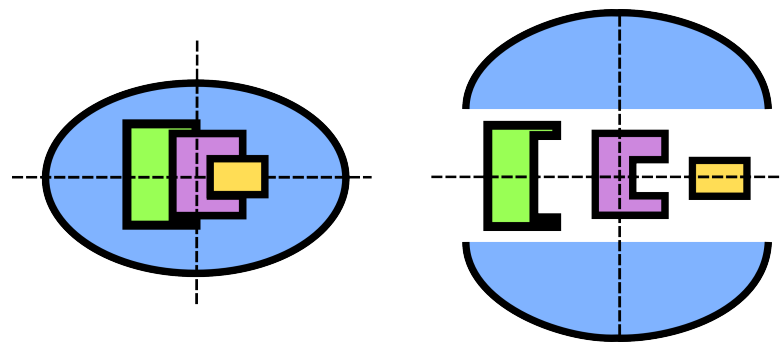
Especially if the graphic conveys technical instructions or how-things-work-descriptions descriptions these axes are usually the directions in which they are assembled [1] or - if applicable - the axis around which they rotate [4]. If the interesting parts are inside a container that is also part of the object, this container is split and its segments are also exploded. In the most simple approach, which is what I implemented, the cutting plane is the normal plane of the explosion direction, with the center of the object's bounding box.

#### Part hierarchy

Take for example an object that has a container and a lot of small parts that inside whose function should be clarified by exploding them in their canonical direction. In this case it might be



**Figure 2.1:** The object of interest is revealed by splitting its container and moving the two segments away from the object in the center.



**Figure 2.2:** The interior parts of an object are exploded in a different direction than their container

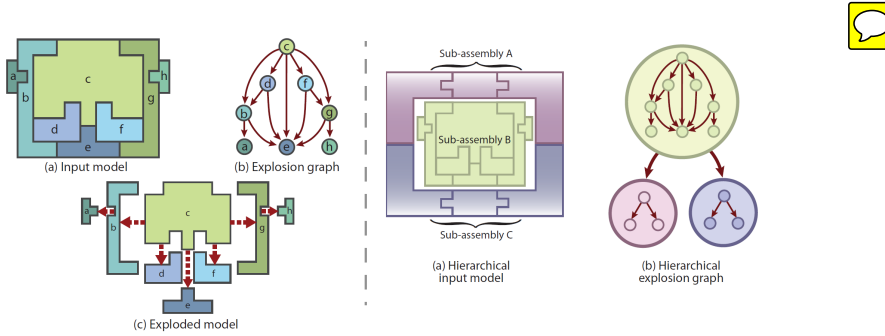
convenient to split and explode the container along a different axis than the internal parts to get a more compact visualization(see 2.2).

In some cases, especially in the case technical visualizations e.g. assembly instructions or visual descriptions of machines the parts would block each other if they are not disassembled in the correct order and direction. A solution for this is an explosion graph representation of the Object [3]: Each node of this directional acyclic graph has edges pointing to parts blocking it from exploding (see 2.3). It is generated by an algorithm, that, starting with all parts in the active set, would recursively check for unblocked parts,remove them from the active set and draw an edge from the unblocked part to any active part that would touch it.

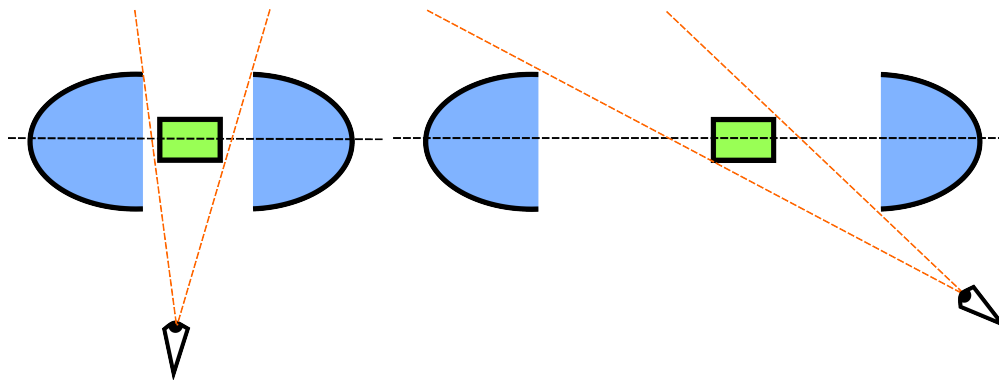
To further expand the possibilities, several parts can be grouped into sub-assemblies that are then part of a hierarchical explosion graph. that has explosion graphs as nodes.

If one now **wanted** to see a certain part of the object of interest in an exploded view, all of the parts that its edges in the explosion graph point to would have to be exploded first. If it were also part of a sub-assembly, all sub-assemblies that would have edges pointing from the part's subassembly would need to be exploded before that.





**Figure 2.3:** Explosion graph and hierarchical explosion graph [3]



**Figure 2.4:** The closer the viewpoint comes to the displacement axis, the larger has to be the offset to reveal the whole object of interest

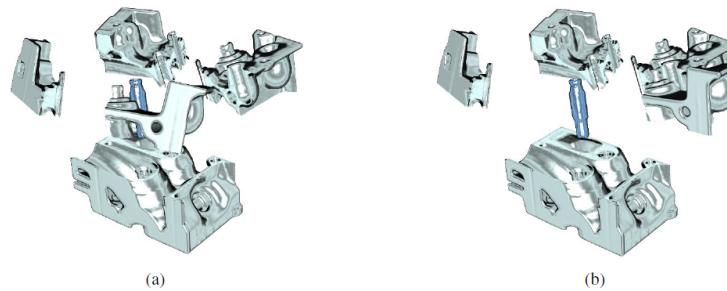
## Visibility

Another challenge is the decision how far to displace the objects. Ideally each part should be fully visible but if there are many objects to be exploded or system the displacement direction is similar to the viewing direction, the size of the graph would grow to be enormous therefore resulting in loss of detail and expressiveness of the visualization. That may make it necessary to have some overlap as a trade-off to retain the compactness of the visualization.

With a freely rotatable and movable viewpoint the user can avoid visual clutter or lack of compactness (depending on how the system behaves) by choosing a viewpoint that provides as little clutter as possible, which narrows down the possibilities of expressive viewpoints to a minimum.

A solution for this would be to use a view dependent force-based displacement behavior as suggested by Bruckner and Gröller [2]: Each exploding part is being displaced by a sum of multiple forces:

- **Explosion force** This force is pushing the part away from its original location, its magnitude is indirectly proportional to  $e^{\|r\|}$  where  $r$  is the distance between the object and the explosion point




**Figure 2.5:** (a) force-based exploded view without viewing force (interesting part is occluded) (b) with viewing force [2]

- **Spacing force** This repulsive force that each exploded part effects on all other exploded parts prevents parts from clustering and is indirectly proportional to  $r^2$  where  $r$  is the distance between the two parts.
- **Viewing force** Additionally, a viewing force is introduced that pushes the parts away from the viewing ray, thus preventing occlusions. It is indirectly proportional to the distance  $r$  between the viewing ray and the part.

This approach results in very intuitive manipulation of the graphic, making the part of the object of interest that one is looking at always visible with smooth transitions between the positions of the parts when changing viewpoints.

In this approach the parts don't have to explode along a predefined axis, the explosion direction can be dynamically determined by the forces, which looks impressive for anatomical data, for a technical illustration it is less desirable, for one of the key advantages of exploded views, the retention of information of a subpart's location in a system is partially lost without a clear displacement direction.

The solution presented in the paper is able to define axes along which to explode, this, though with similarly intuitive interaction by looking at a certain point still has the problem of little oversight when viewing from certain angles. 

## Interactivity

The biggest advantage of a computerized illustration over the classic static illustration is interactivity, most obviously the possibility to view an object from every different direction, which as described above creates its own challenges. Another aspect is the possibility for the user to choose which part of the graphic are interesting to her or him.

In **Li et al.'s** solution [3] this is restricted to the predefined parts of the data sets that can be selected by clicking on them which causes the explosion to adapt, so that the selected part is fully visible. Another possibility is dragging a part along its explosion axis which causes which causes all parts on that axis to move along with it until a part that moves along a different axis is encountered. Also a 3D fish-eye viewing technique can be used that, akin to the viewing

force [2] can be used to move the parts along their axes.

The force-based solution is -due to the use of volume data- more flexible in the definition of parts that can be exploded: the data can be freely partitioned in cuboid subspaces that can be linked by hinges or to an axis, that can be exploded by the forces three above, each of which can be varied between a 0 and a maximum value along with the degree of explosion



## 2.2 Analysis



Since my task required creating exploded views for objects defined by triangle meshes, my approach would be rather like Li et al.'s solution, to use the groups of the mesh as the different parts of the objects which from which an object of interest would be chosen by clicking on it.

This object of interest would stay at its place in the object, while the rest of the object would be split along a cutting plane and then moved in a smooth animation to a point that would leave the object of interest fully revealed. This object would move along the axis defined by the user in a speed that is determined by how much of the Object is visible - the more the object is occluded, the faster the occluding parts move. This non-linear acceleration drew its inspiration from the before mentioned force-based approach to displacing the elements of the object, though my goal was to fully reveal the centered object, which though it normally works, is not guaranteed.

The biggest problem to tackle was now how to handle viewing directions that cause the graphic to grow up to -in th worst case- infinity.

Neither the prospect of visual clutter, indeterminate explosion directions nor the idea of a graphic that would grow infinitely when viewed from the wrong angle proved satisfactory, especially from a usability standpoint: Though the system allows for a freely movable viewpoint, most perspectives just deliver unsatisfactory or even useless results.

Since a maximum threshold for the explosion along an axis was inevitable, my intention was to keep it low enough to retain a satisfactory level of compactness I chose ghosting as a technique to still convey information about the object of interest to the user when it was occluded by another part of the object.

### Ghosting

As the exploded view, ghosting is a smart visibility technique to illustrate the normally occluded inner workings of a complex object. In this case smart visibility is defined like this.

Smart Visibility considers more than just light propagation. For example also the relevance of the individual objects is taken into account. An important object might shine through an otherwise occluding object closer to the viewer. [7]

The advantages of this technique are that the in contrast to the exploded view, objects visualized with ghosting do not take up more space than the objects themselves. Additionally, the inner parts of an object are shown at their actual position inside the object, thus being able to show the composition of an object accurately than an Exploded view might be able to do.

On the other Hand, ghosted views are usually more cluttered because of this, which can lead to

loss of detail and insight. Also they tend to look confusing in case two objects are in between the viewer and the object of interest. To avoid these problems and achieve good results the following rules should be considered: [7]

- faces of transparent objects never shine through
- objects occluded by two transparent objects do not shine through
- transparency falls-off close to the edges of transparent objects



## Practical part

The practical part of my thesis was to create a plug-in for the visualization application “Volumeshop” that was developed at the Computer graphics institute at TU Wien. I built my work upon an Existing plug-in, that split meshes in image space.

### 3.1 Plan and milestone definition:

The practical part was split into three milestones containing the following tasks:

- **Milestone 1** *Selection of split meshes, selected parts are not split and stay in place* Make a simple, intuitive but manual way of creating exploded Views.
- **Milestone 2** *Find a safe distance, find a split plane* Automatize the creation of the visualization, by automatically finding a split plane and an offset.
- **Milestone 3** *Optimize Distance, force-field animation of split, optimize fringe distance cases* Make the visualization more pleasant to look at by adding a seemingly antural force-field animation and prevent unnecessary large offsets by introducing ghosting techniques.

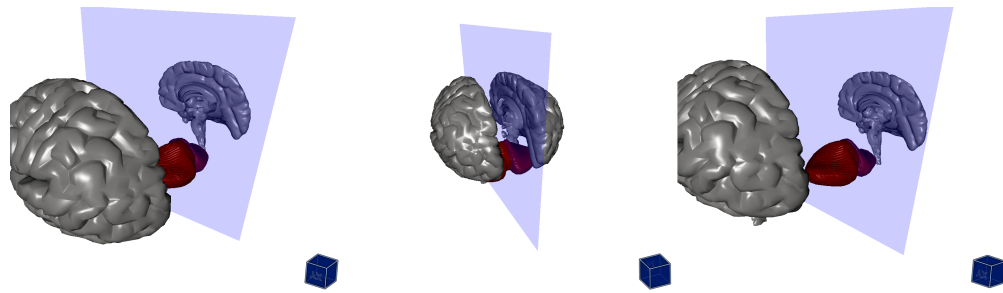
### 3.2 Tools and languages

Since the project is based upon an existing framework, I used its languages, c++ for the main program and the GLSL for the openGL shaders.

### 3.3 Documentation of the implementation each milestone

**Milestone 1: Selection** of split meshes, selected parts are not split and stay in place

The original plug-in drew split meshes by drawing them twice, with a manually defined offset, from a manually defined split plane. The fragment shader then rejects fragments that were be-



**Figure 3.1:** Exploded view of a brain with the offset of the displaced halves are set manually

hind or in front of the **also translated** split plane, rendering them in a predetermined fashion.

The first step towards an exploded view was now to use the already implemented group selection feature to define an object of interest that would not be split or translated like the rest of the mesh. To realize this I introduced a third rendering of the mesh in the display function. This render pass would render only the object of interest in its original location in the non-displaced mesh.

This also required that the “renderMesh”-function be modified, introducing a new boolean parameter “split” that specifies, if the object is rendered in split mode, or in the “object-of-interest-mode”. When the function loops through the groups of a mesh, it checks if the the group is in the group. If that is the case and the function parameter “split” is false, the group will be rendered. If split is false and the group is not selected, it will also be drawn, this time displaced.

Also a new option “split” for the shader was introduced, given that there is no need to pass an offset or split plane to render the object of interest. This basically reverts the fragment shader to the original trianglemesh-shader

After that was done an exploded view of the object was now rendered with manual definition of the offset and split plane with one usability glitch: A colour picking algorithm was already implemented, but it didn’t consider the splitting and translation of the object. This resulted in a behaviour where clicking on a part of the object when it was split would cause false selection or deselection.

To set this right, I modified the the **overlay** function so that it would also be rendered three times like the normal rendering, modifying the “renderGroup” function once more so that it could also render the overlay function and adding an “overlay”option to the shader.

## **Milestone 2: find a safe distance, find a split plane**

The next step would be to automatically place the two halves of the object so that they would not collide with the object of interest and to find a suiting plane to split the object.

As I was aiming for an animated transition between offsets, I chose an implicit approach to finding the ideal offset of the two halves:

Each time the object would be rendered I would first render the three parts (Object of interest, front half, back half) with the low-cost overlay shader counting the rendered pixels of the non-occluded object (*p<sub>unoccluded</sub>*), counting the amount of pixels drawn using OpenGL occlu-

sion queries and during the actual rendering counting the amount of pixels drawn with possible occlusions ( $p_{occluded}$ ). The ratio  $r$  determined by

$$r = \frac{p_{unoccluded} - p_{occluded}}{p_{unoccluded}} \quad (3.1)$$



for

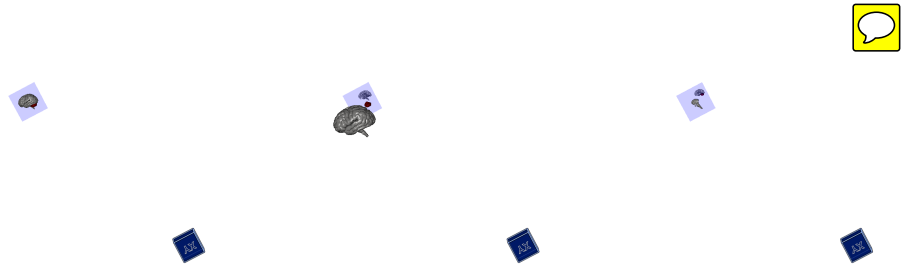
$$p_{occluded} \neq 0 \quad (3.2)$$

is multiplied with the speed specified by user input resulting in the speed at which the offset grows toward an ideal offset which is the maximum offset described in Milestone 3 or an offset of 0 if the difference between ( $p_{unoccluded}$ ) and ( $p_{occluded}$ ) is 0 with the ideal offset different than the maximum offset.

Because the ratio tends towards 0 the more the object is revealed the growth of the offset diminishes the more is revealed, coming to a halt as soon as the whole object is fully revealed which is the moment the ideal offset is set to the current offset.

```
float ratio_selected = occlusions[0]- occlusions[1];
if (ratio_selected>0){
ratio_selected/=occlusions[0];
if (ghosting){
ideal_offset[0]=boundsDiameter;
ideal_offset[1]=-boundsDiameter;
}else{
ideal_offset[0]=20.0f;
ideal_offset[1]=-20.0f;
}
speed=GetPlugin().GetProperty("Speed");
speed*=ratio_selected;
}else{
//stop
}
%
```

This movement is akin to the movement of the object being pulled by a spring toward the point of full revelation of the object of interest, though not a linear spring, because the change of speed is determined by the amount of Pixels that are revealed in each iteration making the spring constant proportional to  $r$ . To avoid that the Object still moves at very low speeds, resulting in huge amounts of unnecessary costly redraws, I introduced a minimum speed  $\epsilon$  so that the object comes to a halt earlier.



**Figure 3.2:** Different objects of interest from roughly the same viewpoint **caus** the offset to grow, placing the object out of view

### Milestone 3: Optimize Distance, force-field animation of split, optimize fringe distance cases

The objective of this last Milestone is to give a smooth appearance to the graphic while the view is changed by User interaction. The first step is to make the transition between two offsets smooth with the transition speed  $s$  proportional to  $\Delta_{offset}$

$$s = s_u \cdot \Delta_{offset} \quad (3.3)$$

thus creating a movement with linear deceleration that comes to a halt when the ideal offset is reached. this behaviour can be observed if the option “Dynamic Offset “ is deactivated.

With dynamic ratio turned on and the object of interest is (partially) occluded the ideal offset is set to the maximum offset and speed  $s$  is multiplied by the ratio  $r$  described in milestone 2 so that when the split parts move away from the object of interest the movement halts when the whole object is fully revealed setting the current offset as the ideal offset.

In case the object is revealed and the explosion needs to be collapsed the ideal offset is set to 0.0 until the object of interest is no longer fully visible, in which case the movement ideal offset is set to maximum and now grows outwards as described before.

This creates a visualization that smoothly adapts to new viewing points and changes in the splitting plane, but has one major disadvantage:

If a plane normal on either side of the plane points approximately in the same direction as the viewing vector, the offset needed to reveal the whole object of interest become very huge in comparison to the object itself. This may prove fatal to the expressiveness of the visualization, given that the goal is to represent an object in context of the parts that are exploded, but the distance between the components is either so large that parts of the components partially move outside of the screen or even completely outside or behind the viewing plane or it is necessary to zoom out or move the camera back so that the whole object is visible causing substantial loss of detail, due to the large offset. If the  $plane\vec{Normal} \cdot viewing\vec{Vector} = \pm 1$  or the object has a certain shape (e.g large at the end in direction of the offset, see 3.2 ) the offset would even grow to infinity.

To circumvent this problem I defined a maximum offset  $o_{max}$  of the objects diameter which is the length of the distance between the minimum and maximum corners of the bounding box of



the mesh. Since the mesh itself has no bounding box, the bounding box has to be accumulated by combining the bounding boxes of the groups of the mesh. This way the distance between the exploded parts can never exceed twice the diameter of the object.

To avoid parts of the object of interest now being occluded I used a simple ghosting technique: The front part, meaning the exploded part that is between the viewer and the split plane, is being rendered translucently if the distance becomes too large. If the current offset  $o_c$  exceeds  $o_{max} \cdot 0.7$  the opacity  $\alpha$  of the front part is linearly interpolated between 1.0 at  $o_c = o_{max} \cdot 0.7$  and 0.5 at  $o_c = o_{max}$  using the formula

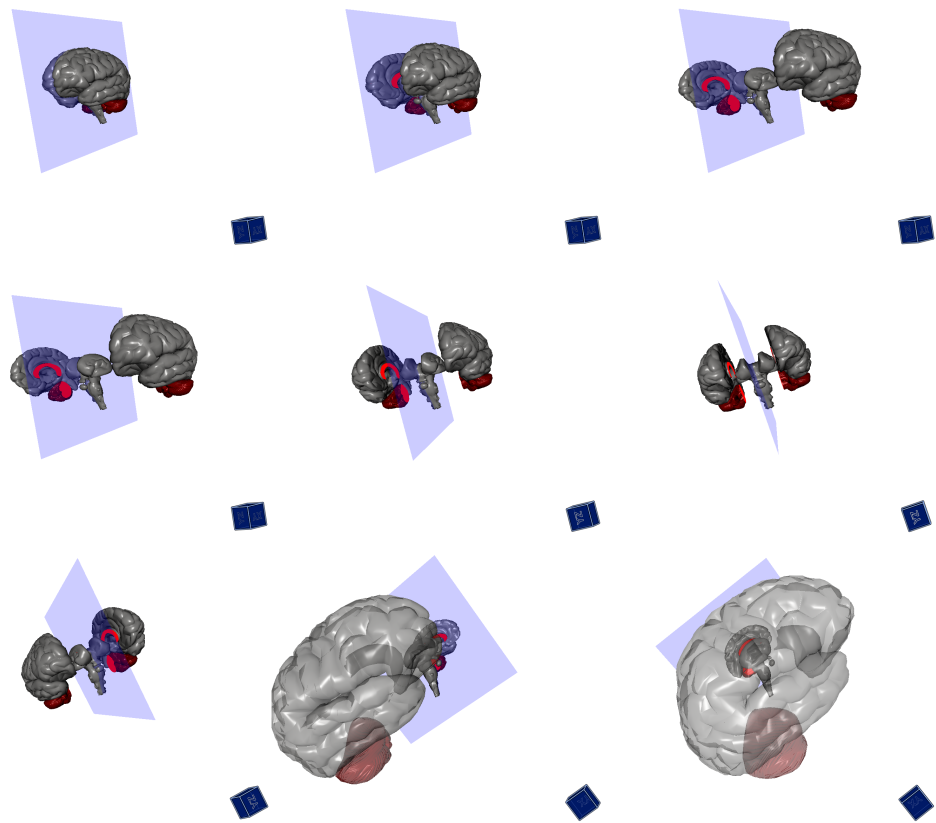
$$\alpha = \frac{o_{max} - o_c}{o_{max} \cdot 0.3} \cdot 0.5 + 0.5 \quad (3.4)$$

This opacity factor  $\alpha$  is then multiplied by  $r$  so that the object stays solid while its not occluding anything and is most translucent if the whole object of interest is occluded completely.

To determine which part is the front part, I calculated the dot product of the viewing vector and the plane normal, using its sign to determine whether the part shifted in direction of the plane normal is the front part or not. This also determines in which order the parts are rendered, with the front part being the last, so that it can be translucently blended over the solid parts.

A problem that now appears is that backface culling is deactivated to so that the cutaway can be rendered correctly, resulting in the translucent objects' backfaces also visible, which causes the graphic to appear slightly confusing and aesthetically unpleasant. This can be avoided by allowing backface culling for a translucent object, if its offset vector doesn't point away from the camera. Because this may be the case if the camera is placed between the original and the translated switch plane, the dot product has to be calculated once more, now for the translated split plane.





**Figure 3.3:** Depiction of the animations and ghosting in the final version of the plugin

## Conclusion

In general I think that the combination of ghosting and exploded views is an improvement upon the technique of exploded views. In comparison to the behavior the pure **explosive** view exhibits when viewed from angles close to the explosion direction it gives more information about the object of interest and it looks better than a solid part blocking it or disappearing from view.

**Ghosting on the other hand loses one of its key advantages, namely that all parts of the object appear in their original location.**

### Future Work

At the moment the explosive view is very basic, with the explosion only along one axis, and a hierarchy that consists of only an object of interest and its context objects. The next steps to improve upon the visualisation would be to introduce more levels into the part hierarchy and devise a way to explode along multiple axes and exploring ways to quasi-naturally animate these behaviours.





# Bibliography

- [1] Maneesh Agrawala, Doantam Phan, Julie Heiser, John Haymaker, Jeff Klingner, Pat Hanrahan, and Barbara Tversky. Designing effective step-by-step assembly instructions. *ACM Trans. Graph.*, 22(3):828–837, 2003.
- [2] Stefan Bruckner and Meister Eduard Gröller. Exploded views for volume data. *IEEE Transactions on Visualization and Computer Graphics*, 12(5):1077–1084, 9 2006.
- [3] Wilmot Li, Maneesh Agrawala, Brian Curless, and David Salesin. Automated generation of interactive 3d exploded view diagrams. *SIGGRAPH 2008*, pages 101:1–101:7, August 2008.
- [4] Niloy J. Mitra, Yong-Liang Yang, Dong-Ming Yan, Wilmot Li, and Maneesh Agrawala. Illustrating how mechanical assemblies work. *Commun. ACM*, 56(1):106–114, 2013.
- [5] Ivan Viola and Meister Eduard Gröller. Smart visibility in visualization. In B. Gooch W. Purgathofer L. Neumann, M. Sbert, editor, *Proceedings of EG Workshop on Computational Aesthetics Computational Aesthetics in Graphics, Visualization and Imaging*, pages 209–216, 5 2005.
- [6] Wikipedia: Illustration. <http://en.wikipedia.org/wiki/illustration>. Accessed: 2013-12-12.

