TECHNISCHE UNIVERSITÄT WIEN - Fakultät für Informatik
Institut für Visual Computing & Human-Centered Technology
Forschungsbereich Computer Graphics

Direct Volume Rendering with Cutting Plane Integration

The exercise example deals with the direct rendering of 3D volume data. The rendering of volume data makes it possible to explore the interior of a data set, depending on the density of the object. This allows data sets to be better understood in their entirety. Interactive editors for creating transfer functions can be used to customize the appearance of the volume.

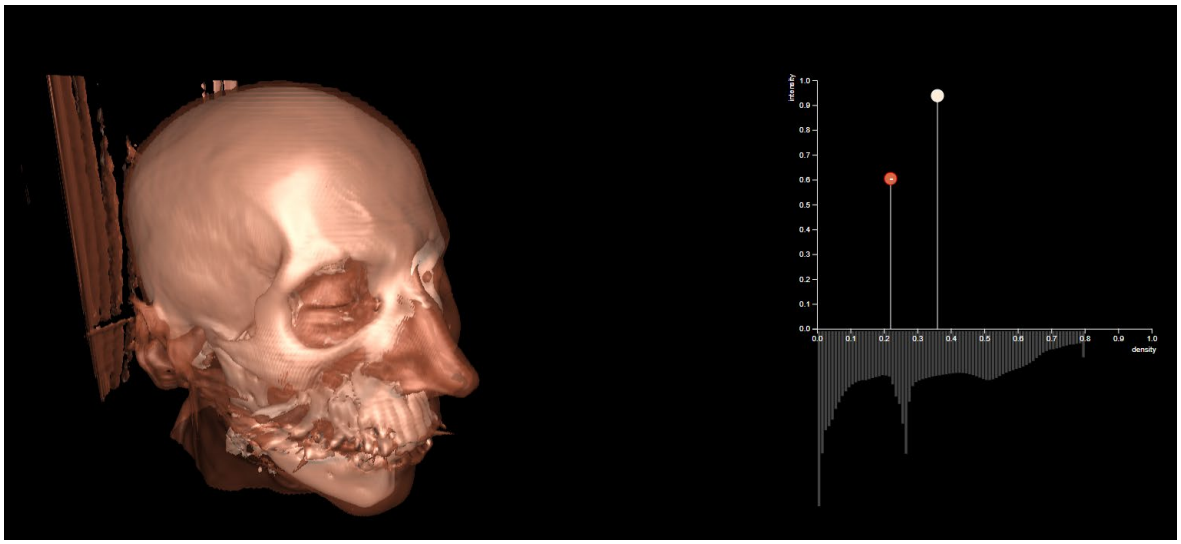


Figure 1: Direct volume rendering of two isosurfaces with a transfer function and density histogram (with Phong Shading).

In this exercise, a simple volume renderer is implemented on the **GPU**, which can display volume data with the help of **raycasting**. Raycasting is a common method to calculate an image from volume data without having to generate geometric primitives (e.g. polygons) from them. You will find more information on raycasting in the lecture units.

Point Distribution

The exercise is worth a total of 44 points. These are distributed as follows:

Minimum requirements (22 points):

- Direct Volume Rendering: visualization of a volume data set using raycasting and any compositing method, e.g. Maximum Intensity Projection **(13 points)**
  = 1. submission
- Visualization of a density histogram **(9 points)**

Please note: if you only implement the minimum requirements, any deduction of points will result in a negative assessment of the entire VU!

Additional Points (22 Points):

- Cutting Plane Integration **(8 points)**
- Interactive editor **(8 points)**

Favoritenstraße 9-11 / E193-02, A-1040 Wien, Austria
Tel. +43 (1) 58801-193201, Fax +43 (1) 58801-193209
www.cg.tuwien.ac.at

- Transfer function **(6 points)**

Framework

We provide you with an **HTML5** / **JavaScript** framework that has already implemented some basic functionalities. The included README.md contains all descriptions of the files. Our framework uses **three.js** (based on WebGL) for 3D rendering and **d3.js** version 6 (based on SVG) for interactive GUI elements and 2D data visualization. WebGL uses the **OpenGL ES Shading Language** (ESSL). You can find a quick reference here:

https://www.khronos.org/opengles/sdk/docs/reference_cards/OpenGL-ES-2_0-Reference-card.pdf

Our framework loads shader files at runtime and must therefore be started from a server. We recommend using the development environment WebStorm from JetBrains. If you enter your TU e-mail address, you can download a free version of the IDE here:

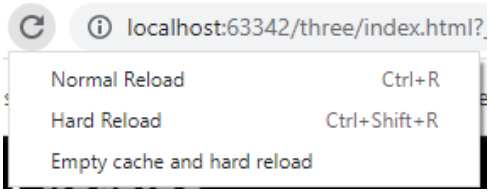
https://www.jetbrains.com/community/education/#students. Create a new WebStorm project from the folder of your framework. Now open the file index.html. In the top right corner, you will find a list of browser icons. This starts your project from a server in the desired browser.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Vis1</title>

    <link rel="stylesheet" type="text/css" href="style.css">

    <!-- include THREE.js -->
    <script src="three.js/build/three.js"></script>
```

Most browsers have helpful Developer Tools integrated. In Chrome, for example, you can open the **Developer Tools** with the shortcut **Ctrl+Shift+i**. Here you can inspect DOM elements, see your console output and any error messages (including compile errors from shaders), and also debug your code. You can find an overview of Chrome Developer Tools here:

https://developer.chrome.com/docs/devtools/overview/

Especially when working in the shader, it is advisable to always have the console open. This makes it possible to perform a "Hard Reload" after changes in the source code and to reload all files.

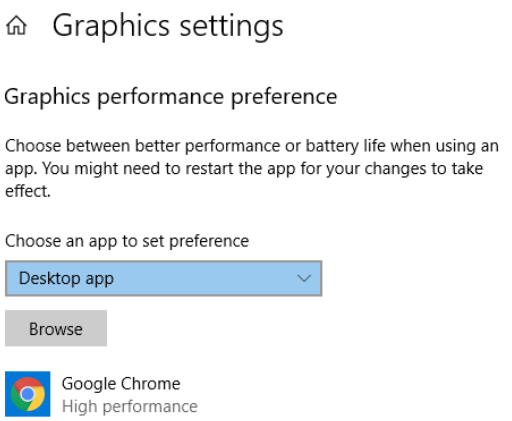


Make sure that your browser is running with GPU support. You can check this using the Task Manager:

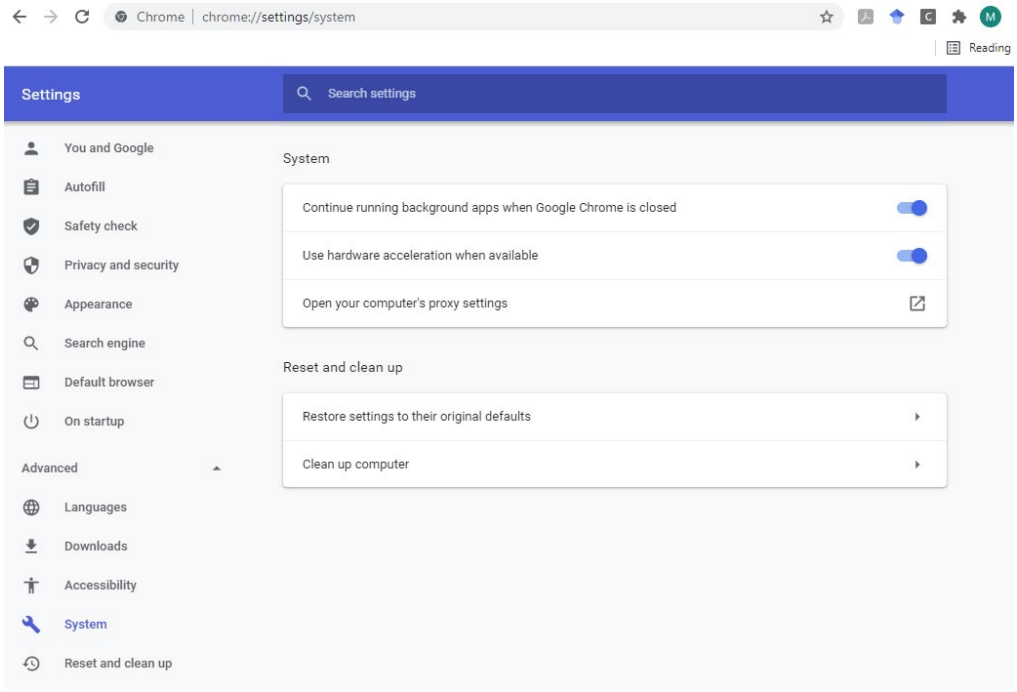
Name	Status	11% CPU	59% Memory	0% Disk	0% Network	5% GPU	GPU engine
Apps (12)							
Google Chrome (12)		0,2%	688,2 MB	0,1 MB/s	0 Mbps	0%	GPU 1 - 3D

Favoritenstraße 9-11 / E193-02, A-1040 Wien, Austria
Tel. +43 (1) 58801-193201, Fax +43 (1) 58801-193209
www.cg.tuwien.ac.at

For Windows laptops, selected applications can be selected for improved graphics performance if necessary:



In addition, you should ensure in the Chrome settings that Chrome is actually using hardware acceleration:



In Chrome, you can measure the frame rate using an integrated FPS counter. To do this, open the console (Ctrl+Shift+i) and then enter Ctrl+Shift+p and type "FPS Counter" in the search window.

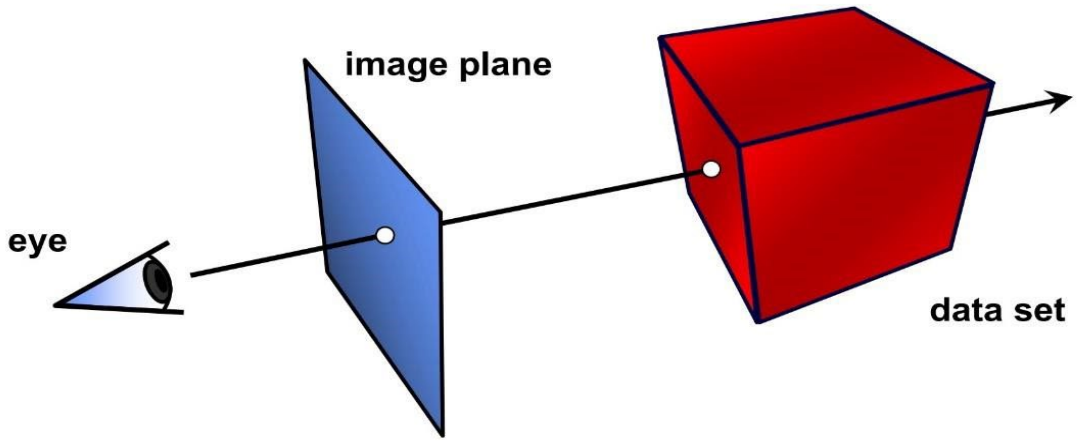
The framework has been tested for Chrome.

Delivery

Please note the following points before submitting:

- Upload your extended framework without the data sets as a .zip file.
- Please remove any temporary files beforehand.
- Test and debug your program thoroughly before submission!
- Please also add the interaction options you have added to the README.md!

1. Direct Volume Rendering* (13 Points)



To create an image using raycasting, one ray per pixel is cast through the image plane. The ray is intersected with the volume. The volume data is arranged in a Cartesian 3D grid. Voxel information is stored at the grid positions (density values). In the data provided, the density values assume values in the **range [0,1]**.

The density is read out at regular positions (samples) along the line of sight and, depending on the method used, a value is determined for the current beam. You will become familiar with various methods for volume visualization in the lecture. One simple method, for example, is **Maximum Intensity Projection (MIP)** - see Figure 2 and description below.

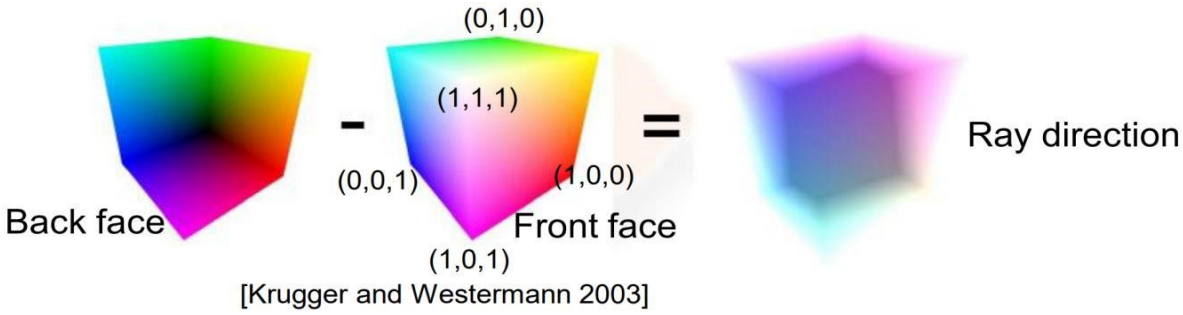
Figure 2: Maximum Intensity Projection.

Procedure:

The basic idea of GPU-based raycasting is that the entire volume is stored as a 3D texture on the GPU and rays are shot through the volume in a fragment shader to calculate the respective pixel value. Each pixel corresponds to a ray. There is a Two-Pass implementation and a more efficient Single-Pass implementation.

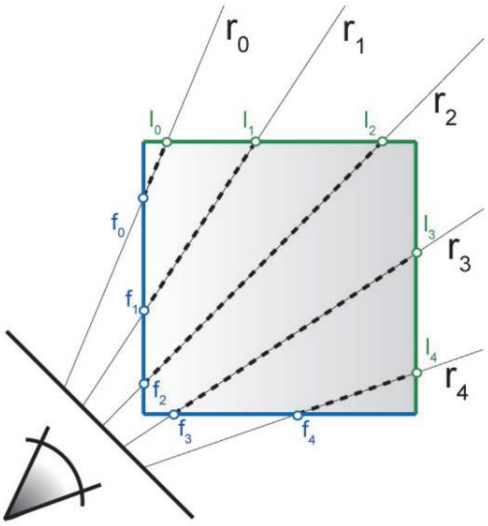
Two-Pass Raycasting:

A method to get to the start and end points of the rays is to render the front faces and back faces from the bounding box of the volume and save them in an FBO.



[Krugger and Westermann 2003]

If you subtract the coordinates of the front faces from the coordinates of the back faces, you get the direction of the shot rays. The 3D coordinates of the volume are illustrated above in RBG.



The next step is to carry out the raycasting. To start raycasting, a rectangle should be rendered that covers the entire image area. During raycasting, the density values are then scanned in regular steps along the rays. (In the image: start at $f0$ -$f4$ and end at $l0$ -$l4$.) The sampling rate influences the performance and quality of the resulting image, whereby a suitable value is to be calculated by the students.

Single-Pass Raycasting:

Two-Pass Raycasting requires two render passes and is therefore not optimal. It is also possible to perform raycasting in a single render pass. To do this, the bounding box of the volume is rendered as a simple box geometry. In contrast to Two-Pass Rendering, the camera position and the dimension of the bounding box must also be forwarded to the shader. The ray can now be defined by the position of the camera (origin) and the direction from the camera to the fragment.

As this is an axis-parallel bounding box, the intersection points between the ray and the bounding box of the volume can be calculated efficiently using the so-called "Slab Method". You can find an

explanation here, for example: https://tavianator.com/2011/ray_box.html. As soon as the intersection points are known, the beam can be scanned as described above.

Maximum-Intensity Projection (MIP):

With raycasting, the density is read out at regular positions (samples) along the line of sight, and, depending on the method used, a value is determined for the current ray. MIP is one of the simplest methods for both Single-Pass and Two-Pass raycasting. For each sample, it is checked whether the current density value corresponds to the maximum along the ray - the highest density value along the ray ultimately determines the color value for the fragment.

Notes on implementation:

We use three.js for rendering. In the framework, volume data is loaded with the format provided by us and saved as a float array (see volume.js). You can use THREE.Data3DTexture to create the 3D texture.

For rendering the bounding box coordinates (Two-Pass Raycasting) and raycasting, you must use appropriate shaders. You can derive from the provided class Shader (shader.js) and specify the corresponding shader files or set the uniforms. You can use the dummy example testShader.js as an example for use.

You can simply read the current camera position (Single-Pass Raycasting) from the existing camera in vis1.js (camera.position).

Summary:

1. One of the two methods to generate the beam:
   a. Two-pass rendering: Generation of the front faces and back faces: Render the coordinates of the front faces and back faces into one texture each
   b. Single-pass rendering: Calculation of intersection points of camera ray with volume
2. Raycasting: use the start position of the front faces and scan the volume along the ray direction
3. Use a compositing method (e.g. MIP) to calculate the color values

Raycasting (two-pass or single-pass rendering) with MIP is part of the minimum requirements for a positive finish.

2. Density Histogram* (8 + 1 Points)

A histogram should be provided to help users understand the distribution of density values in the volume dataset. The histogram should visualize the distribution of density values present in the loaded volume dataset, making it easier to analyze the dataset's characteristics. The histogram should be updated whenever a new dataset is loaded, and transitions should be animated for better readability.

The histogram can also help you select a sensible iso-value for the Transfer Functions (see task 5). It should serve as an extension of the editor (see below), as illustrated in Figure 3.

Make sure that the histogram is recalculated when a new data set is loaded. The transitions should be animated.

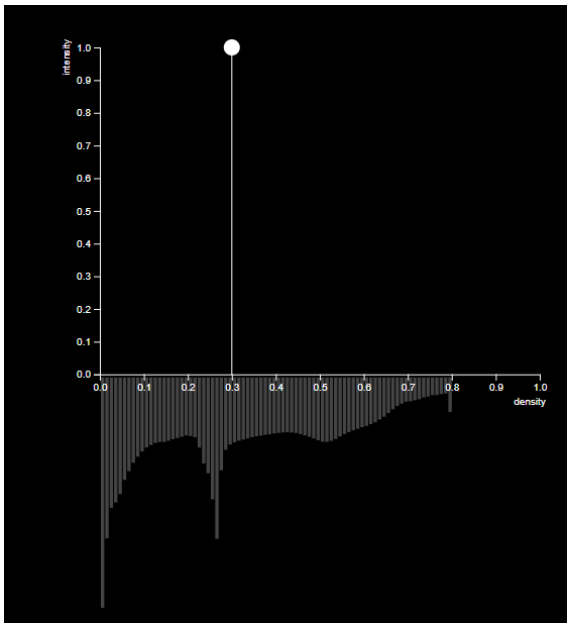


Figure 3: Editor with Histogram.

Notes on implementation:

There are corresponding auxiliary functions in d3-array for calculating histograms. Please note the correct creation and updating of the stored data using d3 data joins (see https://github.com/d3/d3-selection), which also support animated transitions. You should also find an easily readable scaling of the histogram bars. D3 offers a variety of scaling methods in the d3-scale package.

Summary:

- Easy-to-read histogram for the loaded volume dataset (**8 points** - part of the minimum requirements for a positive conclusion)
- Animated transitions (**1 point**)

3. Cutting Plane Integration (8 Points)

This task involves adding a ***cutting plane*** to the raycasting process, enabling users to reveal the internal structures of the volume. The cutting plane should act as a **hard boundary**, ensuring that only the selected portion of the volume is visible based on the plane's position. The implementation should correctly modify the raycasting calculations to support volume clipping.

Summary:

- Integrate a **cutting plane** into the raycasting process to reveal internal structures: **8 points**.

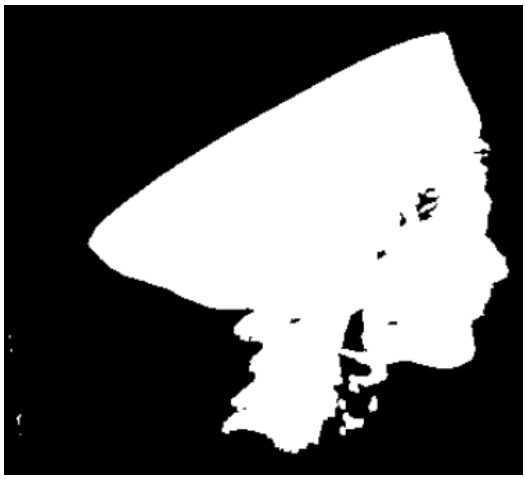


Figure 4: *Cutting plane integration.*

Figure 5: Cutting plane with shading (see Bonus Points below).

4. Interactive Editor (8 Points)

Once the cutting plane is integrated, the next step is to make it **interactive and customizable**. The editor should allow users to **manipulate the cutting plane in real-time**, including adjusting its **position, rotation, and visibility settings**. The cutting plane should, therefore, be interactive, allowing users to translate or rotate it and toggle between rendering "Above" or "Below" the plane.

The application should be extended by an interactive editor that allows the user to manipulate the cutting plane interactively. The editor does not have to look like Figures 1 and 3, but should at least have the following features:

- Translation and rotation controls for adjusting the cutting plane in real-time.
- Toggling between rendering "Above" or "Below" the plane (integrated within the controls).
- Interactive selection of the cutting plane's color (at least 10-20 systematically selectable colors, including white).

Additionally, the density histogram should dynamically update in real-time to reflect the voxels visible through the cutting plane. This requires ensuring that the histogram recalculates whenever the cutting plane changes and displays only the density distribution of the currently visible voxels.

Summary:

- Interactive transformation (translation, rotation, and orientation adjustments) for the cutting plane (**4 points**).
- Interactive color selection for the cutting plane (**2 points**).
- Dynamic histogram updates (update the histogram dynamically to reflect visible voxels based on the visible volume): 2 points.

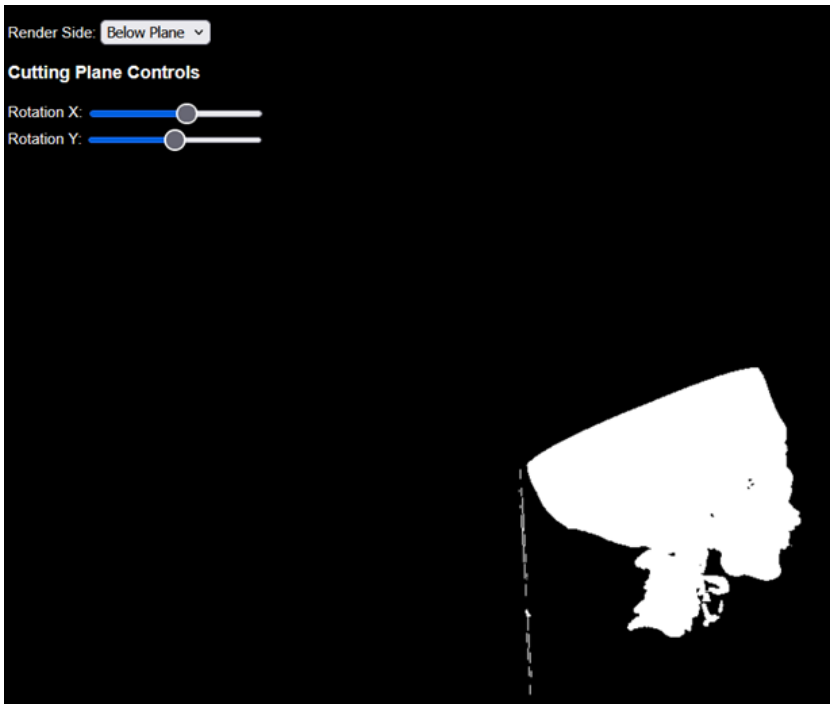


Figure 6: *Cutting plane integration and user controls on the left.*

Notes on implementation:

We use **d3.js v6** for drawing and event handling of the editor. We recommend creating at least one dedicated class or file for this part of the exercise. D3 offers some helpful functions for this task: axes can be created with the help of *d3-axis*, for example.

Documentation for event handling can be found here: https://github.com/d3/d3-selection#handling-events. Attention: with version 6 the event handling in d3 has been slightly changed. Tutorials and examples based on older d3 versions may no longer be compatible.

To draw the color picker, you can use *d3-color*, which already contains many methods for color management. The HTML5 color picker can also be used (see https://www.w3schools.com/colors/colors_picker.asp).

5. Transfer Function (6 Points)

Transfer functions are used to map certain functional areas (areas of density values) to certain materials (color values, opacity). Such transfer functions can be used, for example, to determine that density values that are typical for bone are given a different color than those that are typical for tissue. The editor should allow at least two isosurfaces to be added and deleted. For each surface, it should be possible to change the iso values and colors (as defined in task 5) as well as the transparency of the surface. Shading is not required, but additional points may be awarded for its implementation (see 6. Bonus below).

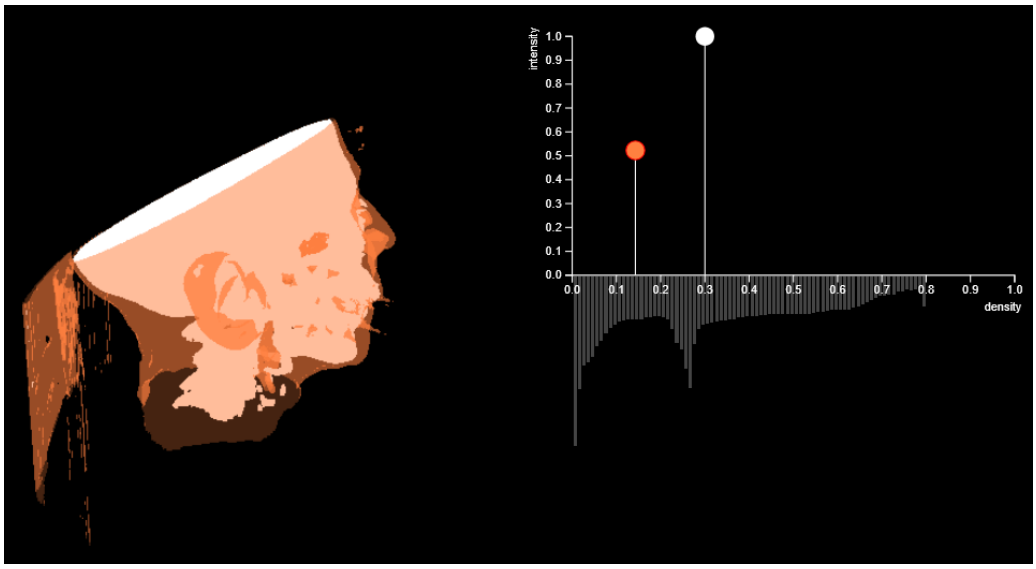


Figure 7: Visualization of isosurfaces using a transfer function, with corresponding color and opacity mapping based on density values.

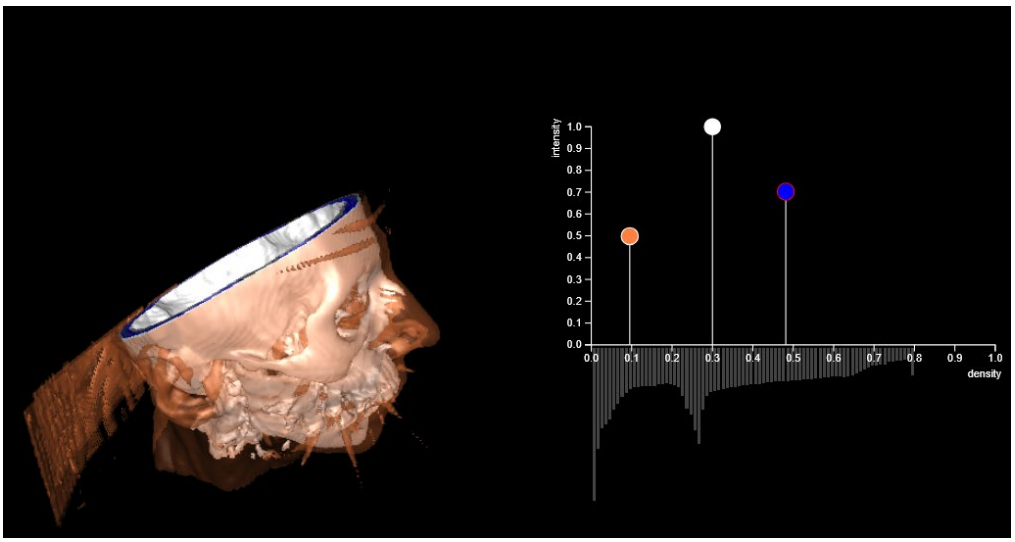


Figure 8: Transfer Function implementation.

The editor should allow at least two isosurfaces to be added and deleted. For each surface, it should be possible to change the iso values and colors (as defined in task 4) as well as the transparency of the surface.

Favoritenstraße 9-11 / E193-02, A-1040 Wien, Austria
Tel. +43 (1) 58801-193201, Fax +43 (1) 58801-193209
www.cg.tuwien.ac.at

Notes on implementation:

The interaction method for adding and deleting is up to you. For example, it would be possible to use modifier keys or menus. Please document the implemented interaction options in README.md.

6. Bonus (maximum 5 bonus points)

You can incorporate extremely attractive enhancements relatively easily, for example**, Phong Shading, Ambient Occlusion**, **First-Hit Compositing or Alpha Compositing** as an alternative compositing method.

In total, a maximum of 5 bonus points can be achieved for the exercise.

Notes on First-Hit Compositing

With First-Hit Compositing, an iso-value is set that defines a surface. During raycasting, it is therefore checked in each step whether the selected iso value is between the density value of the current position and the density value at the next sampling step. If this condition is met, linear interpolation is performed between the two sampling positions to calculate the actual position of the first hit. In other words, an interpolation factor of the iso value between the two density values is determined.

This interpolation factor is then used to interpolate the 3D positions of the sampling steps. Figure 9 shows the sampling positions of the first hit.

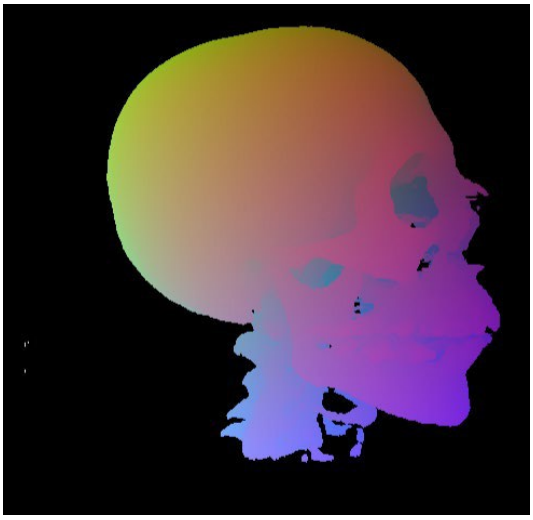


Figure 9: First-Hit positions (Iso-Value: 0.3).

Notes on Gradients and Shading

To make the surface of the object more recognizable than in Figure 9, the gradient of the surface at the First-Hit position should be determined and used as a normal for shading. The gradient describes the direction of the greatest change, i.e., it points away from the surface and is calculated as follows:

$$\nabla f\begin{pmatrix}x\\y\\z\end{pmatrix} = \frac{1}{2}\cdot\begin{pmatrix}f(x-\varepsilon)-f(x+\varepsilon)\\f(y-\varepsilon)-f(y+\varepsilon)\\f(z-\varepsilon)-f(z+\varepsilon)\end{pmatrix}.$$

Tip: To check the correctness of the normals, they can be output as RGB values in the same way as the positions.

A simple shading (e.g., Phong Shading or Blinn-Phong Shading) with any light direction can be calculated using the normals of the surface. Students are free to choose the parameters, but care should be taken to ensure that the rendered surface is recognizable.

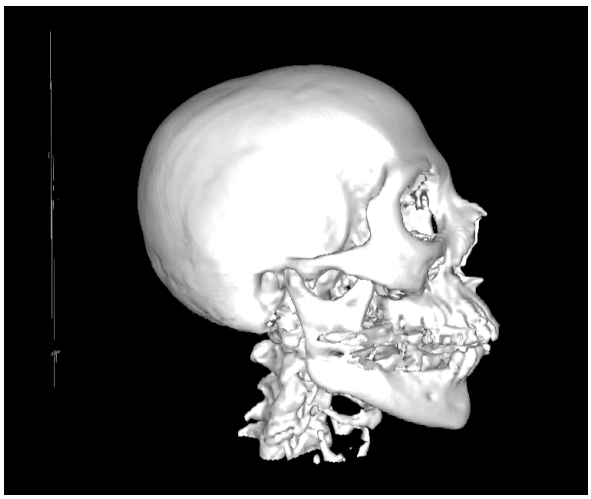


Figure 10: Illuminated Iso-Surface (Iso-Value: 0.3).

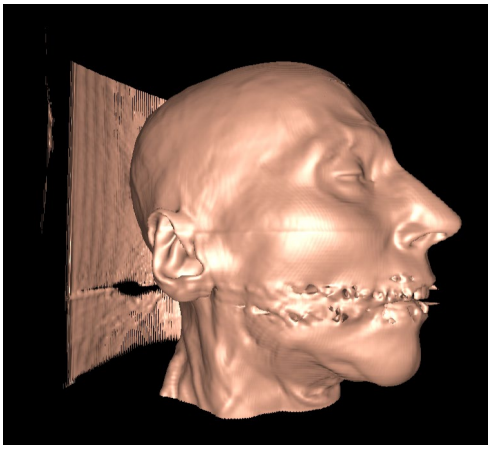


Figure 11: Illuminated iso surface with assigned surface color (iso-value: 0.2).