

Lab Class Visualization

Part 1: Assignments InfoVis: *10 Pt.*

(Patrick Riehmann)

Part 2: Assignments SciVis: *10 Pt.*

(Carl-Feofan Matthes)

Part 3: Final Project: *30 Pt.*

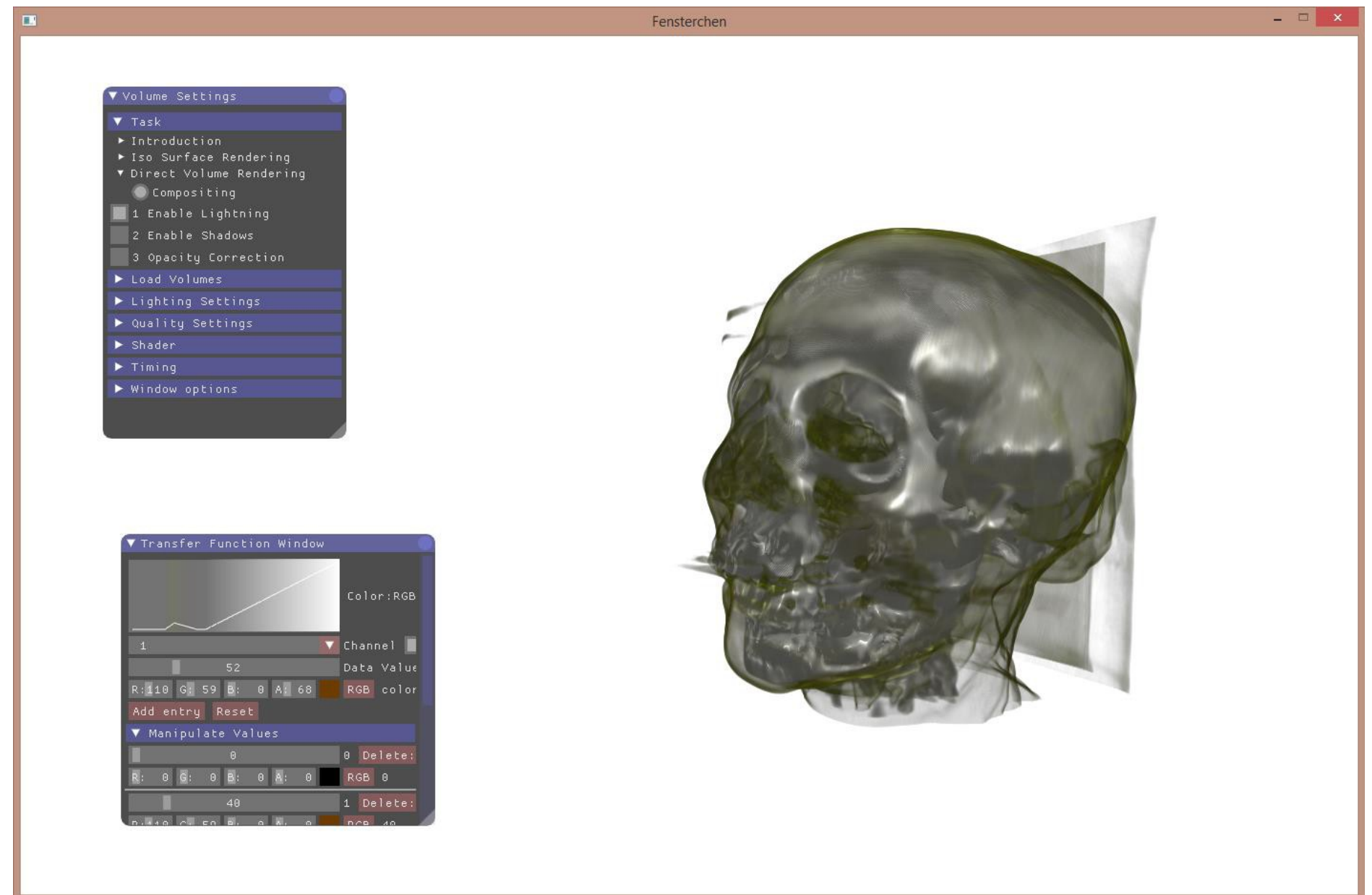
Final Project (SciVis / InfoVis)

- Topic: up to you (either InfoVis or SciVis)
- Expenditure of time: ~40h/Student
- Final Project Submission:
Short description + source code
Final project deadline: **September 10, 2018, 12 pm**

Send to
patrick.riehmann[at]uni-weimar.de
carl-feofan.matthes[at]uni-weimar.de
- Project presentation: **September 12, 2018, 10 am, DBL top floor**

SciVis-Assignments

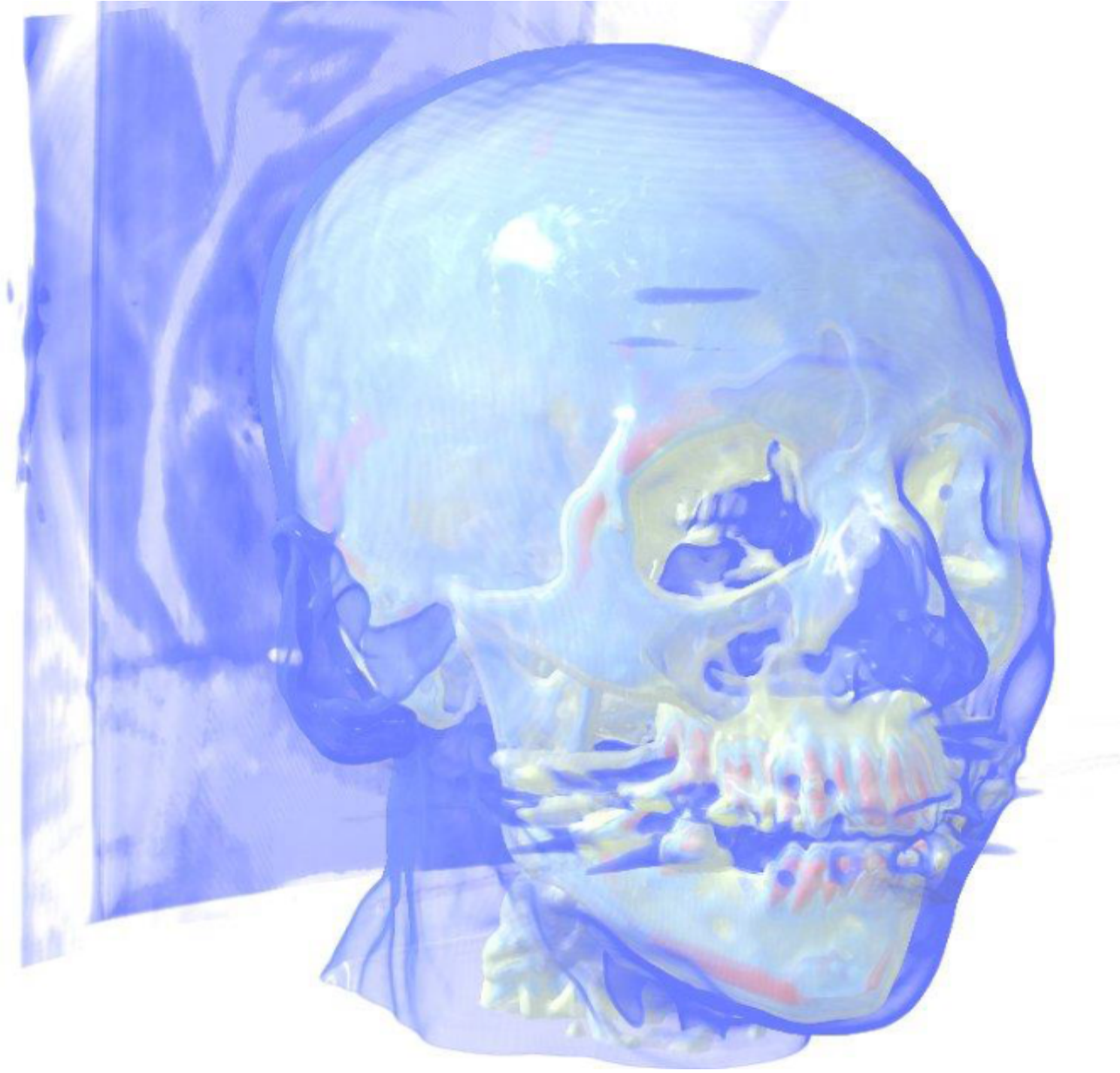
- **10 tasks, 10 points**
- Volume raycasting concepts
- Modus operandi
GLSL, C++11, OpenGL, Cmake
- Raycaster is implemented in GLSL
- No need to touch any C++ files
- Submission:
Lab class on Juli 3, 2018
Lab class on Juli 10, 2018



Backup your code!

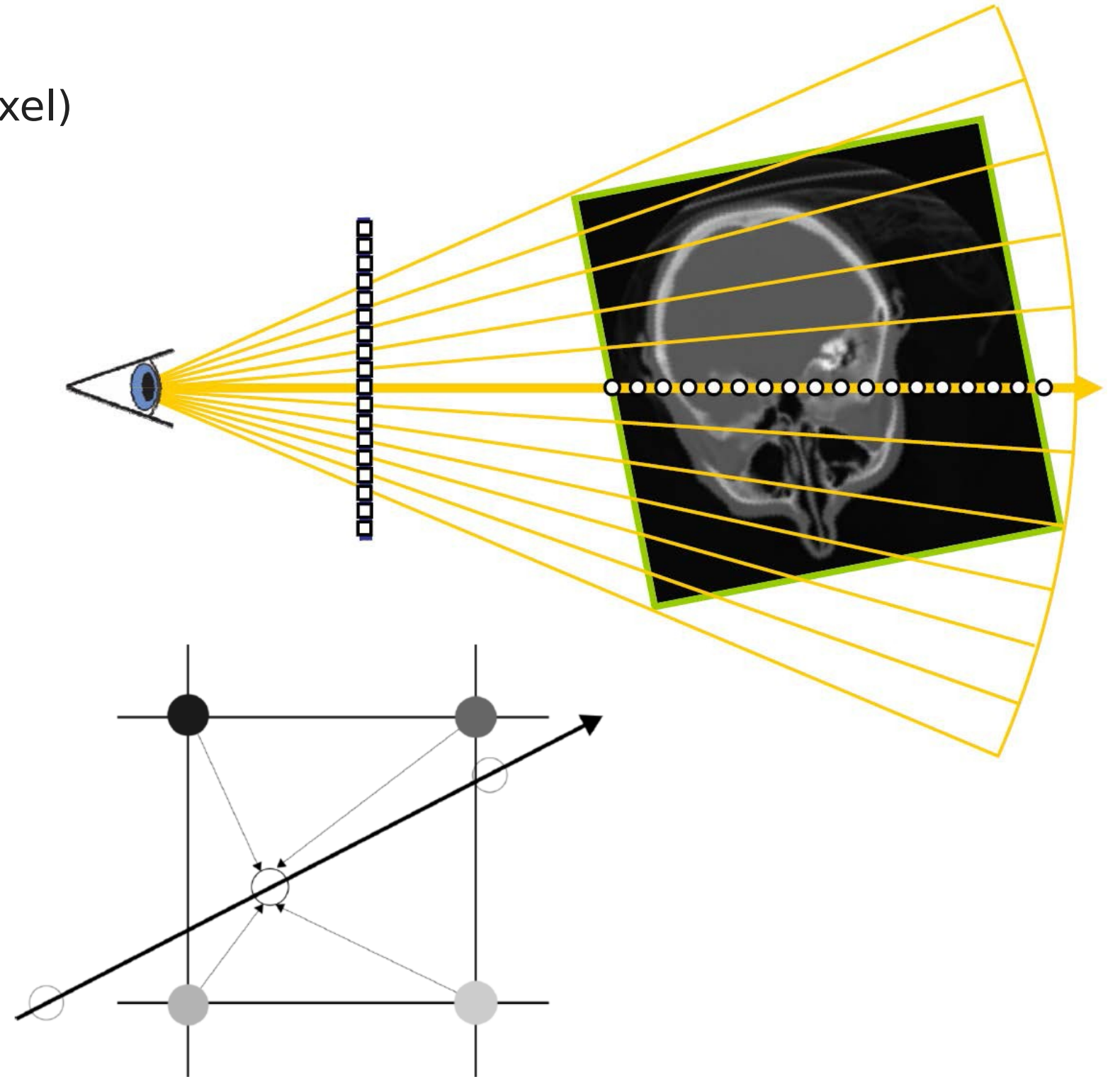
- **It is your own responsibility to backup your code!**
- Use *github, dropbox, USB-stick, your phone...*
 - whatever you feel comfortable using
- If you cannot run your code on the day of submission, you will fail the lab class!

GPU-based Volume Raycasting



GPU-based Volume Raycasting

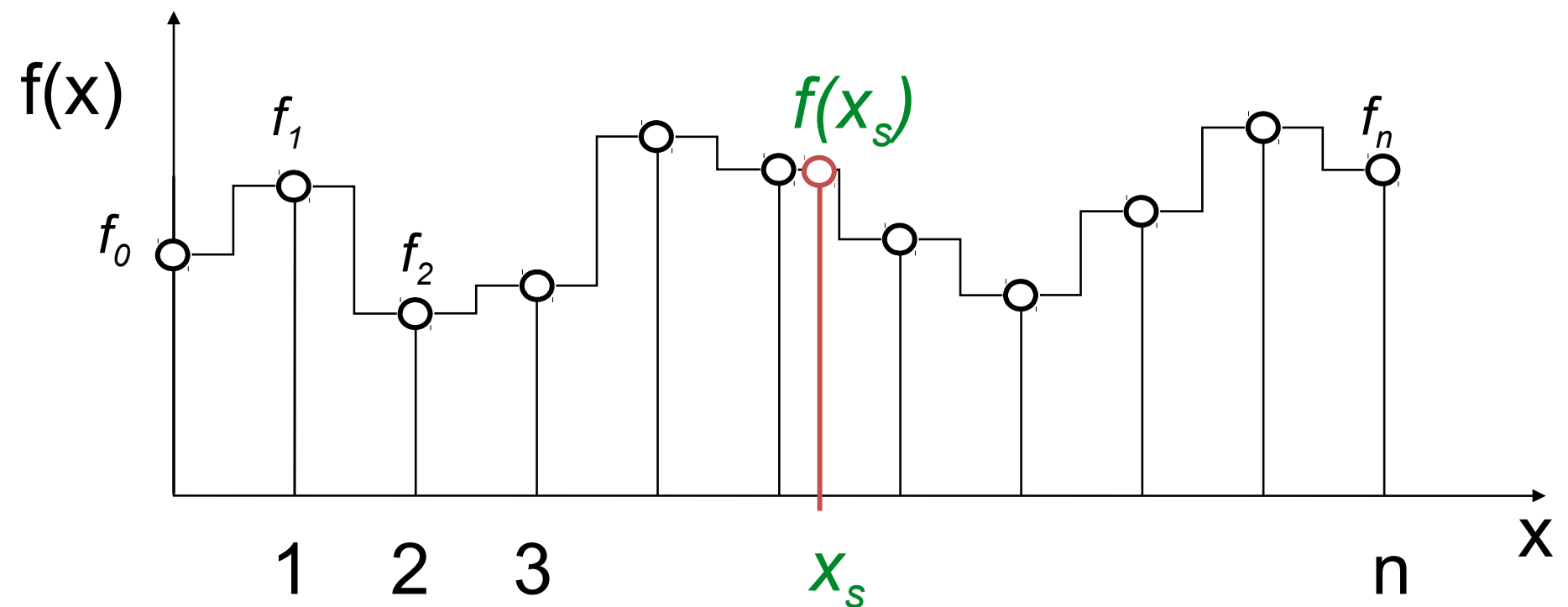
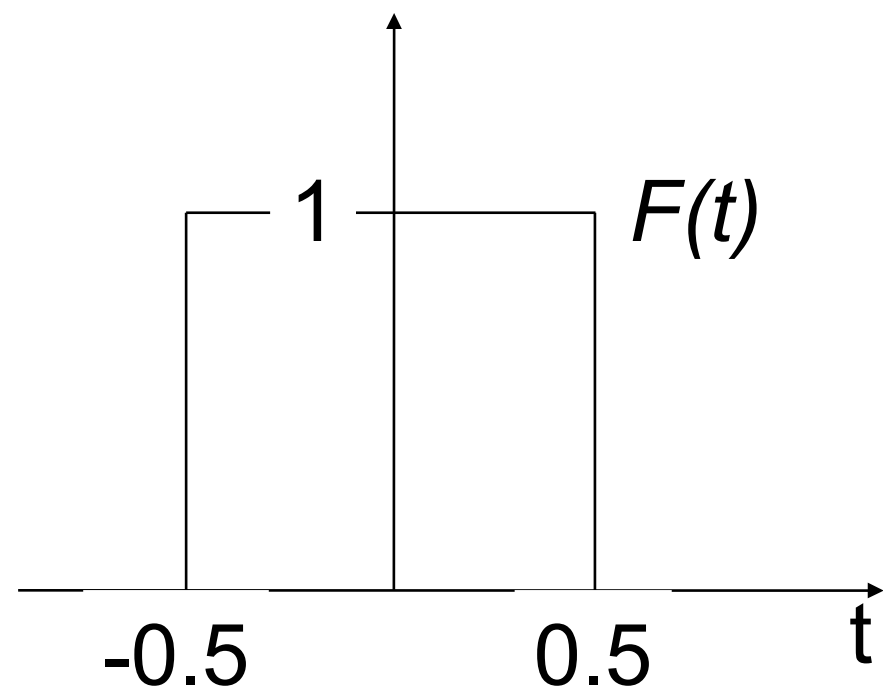
- The **volume** is stored in a 3D-Texture (1 byte per voxel)
- A **ray** is generated from each fragment
- The volume is **sampled** along each ray
- E.g. **Maximum Intensity Projection**:
Choose the maximum of all samples along a ray to color the pixel
- Data values defined on a **regular grid**
- Sampling the volume requires **interpolation**



Nearest Neighbor Interpolation

- **Box-shaped** reconstruction kernel $F(t)$

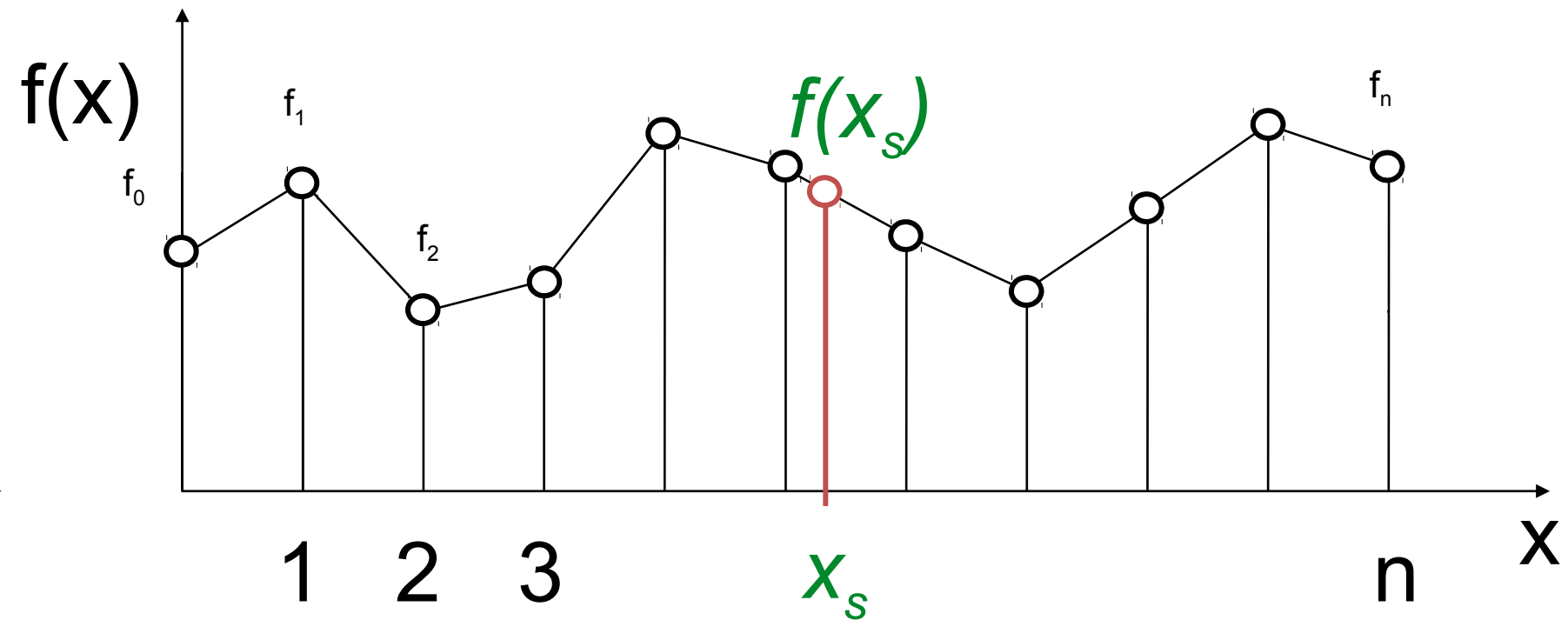
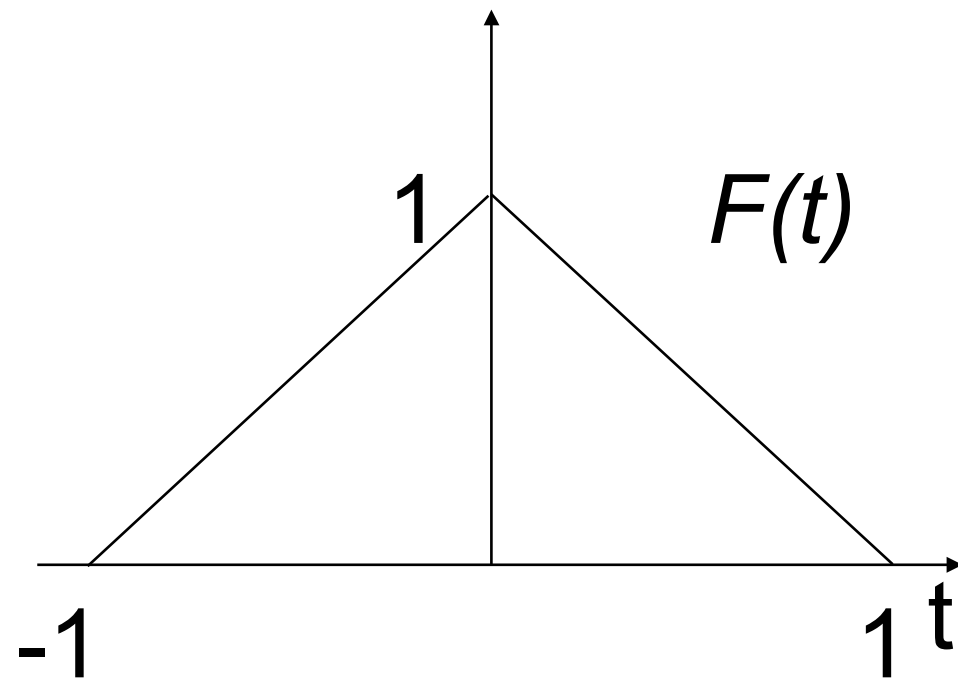
$$f(x) = \sum_{i=0}^n f_i F(x - i)$$



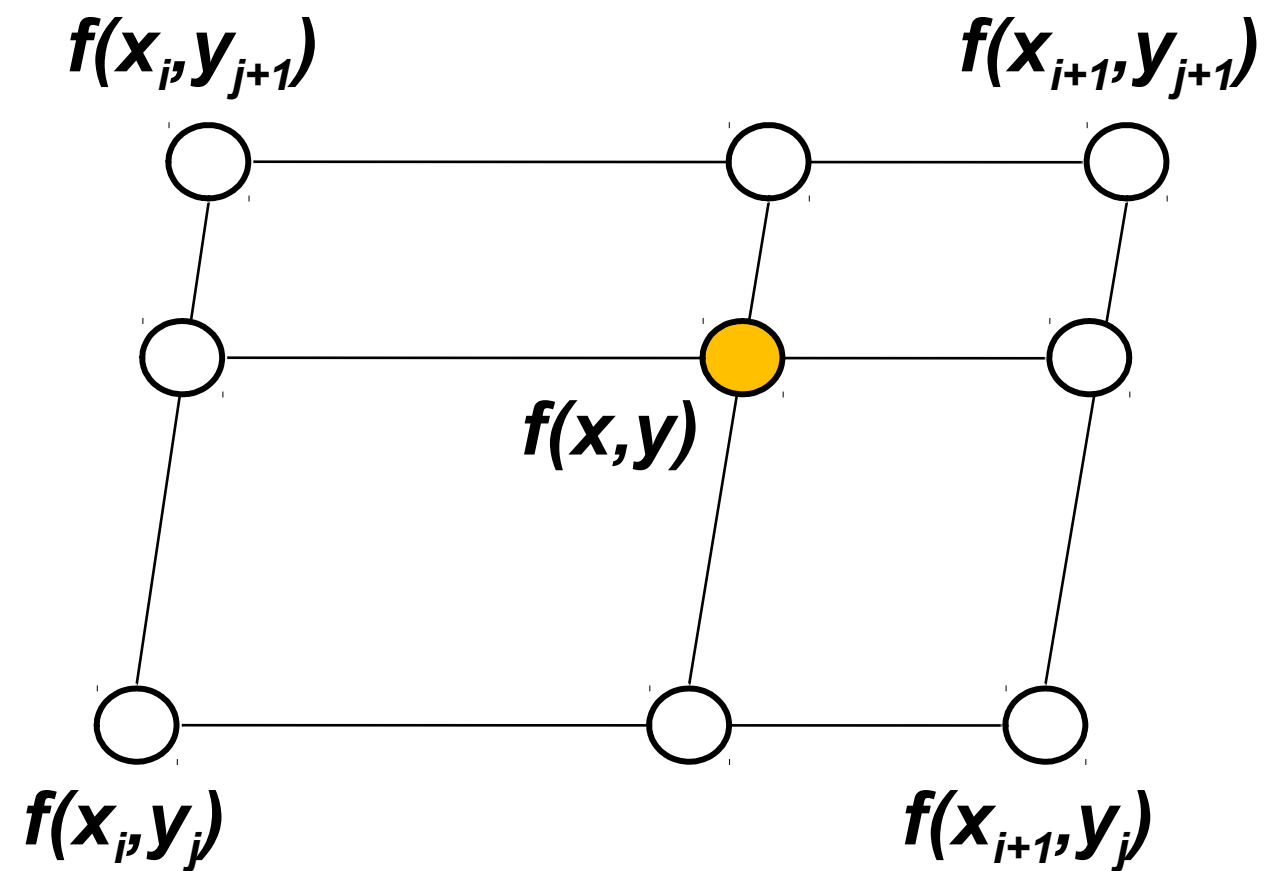
Linear Interpolation

- **Tent-shaped** reconstruction kernel $F(t)$

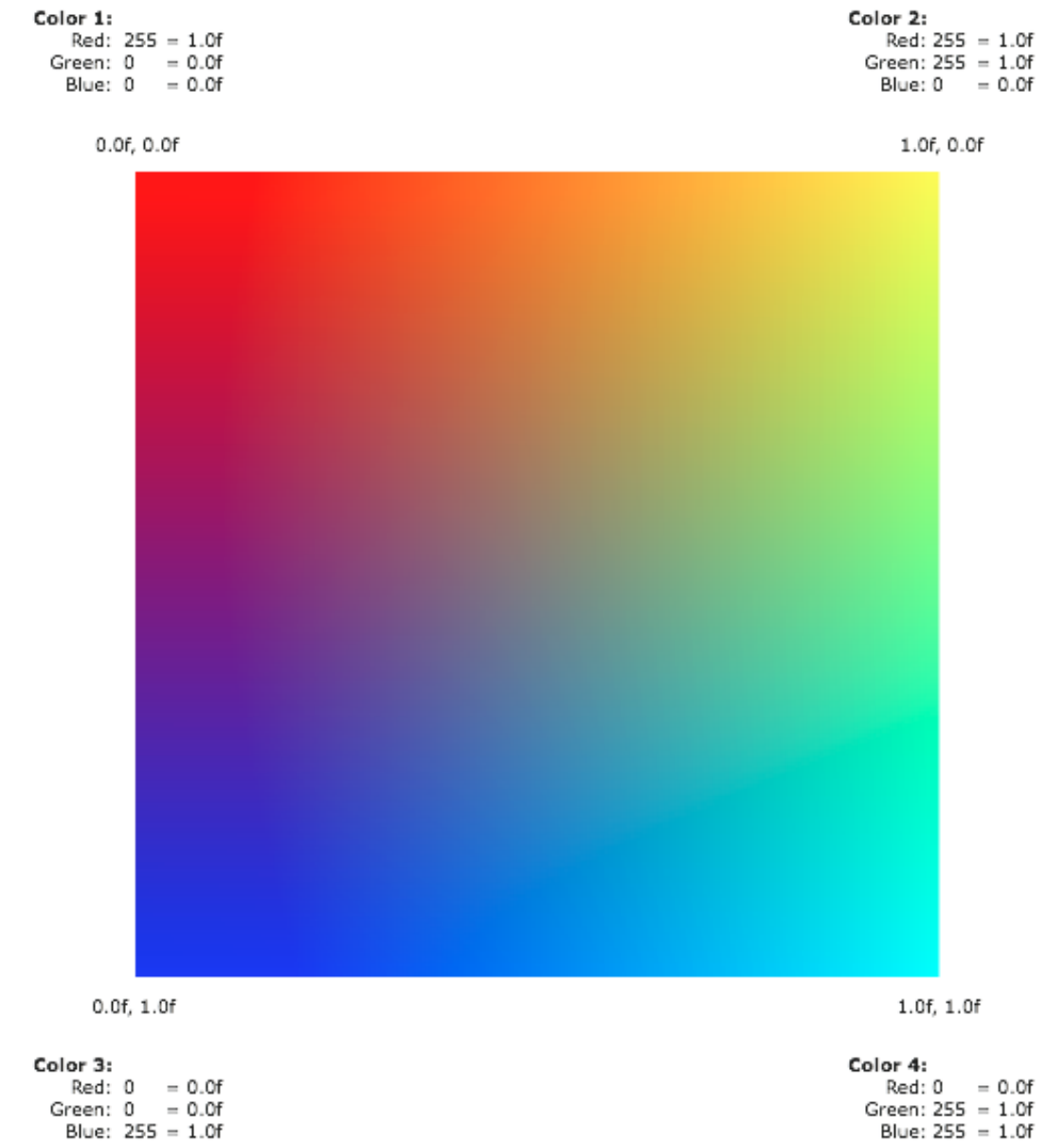
$$f(x) = \sum_{i=0}^n f_i F(x - i)$$



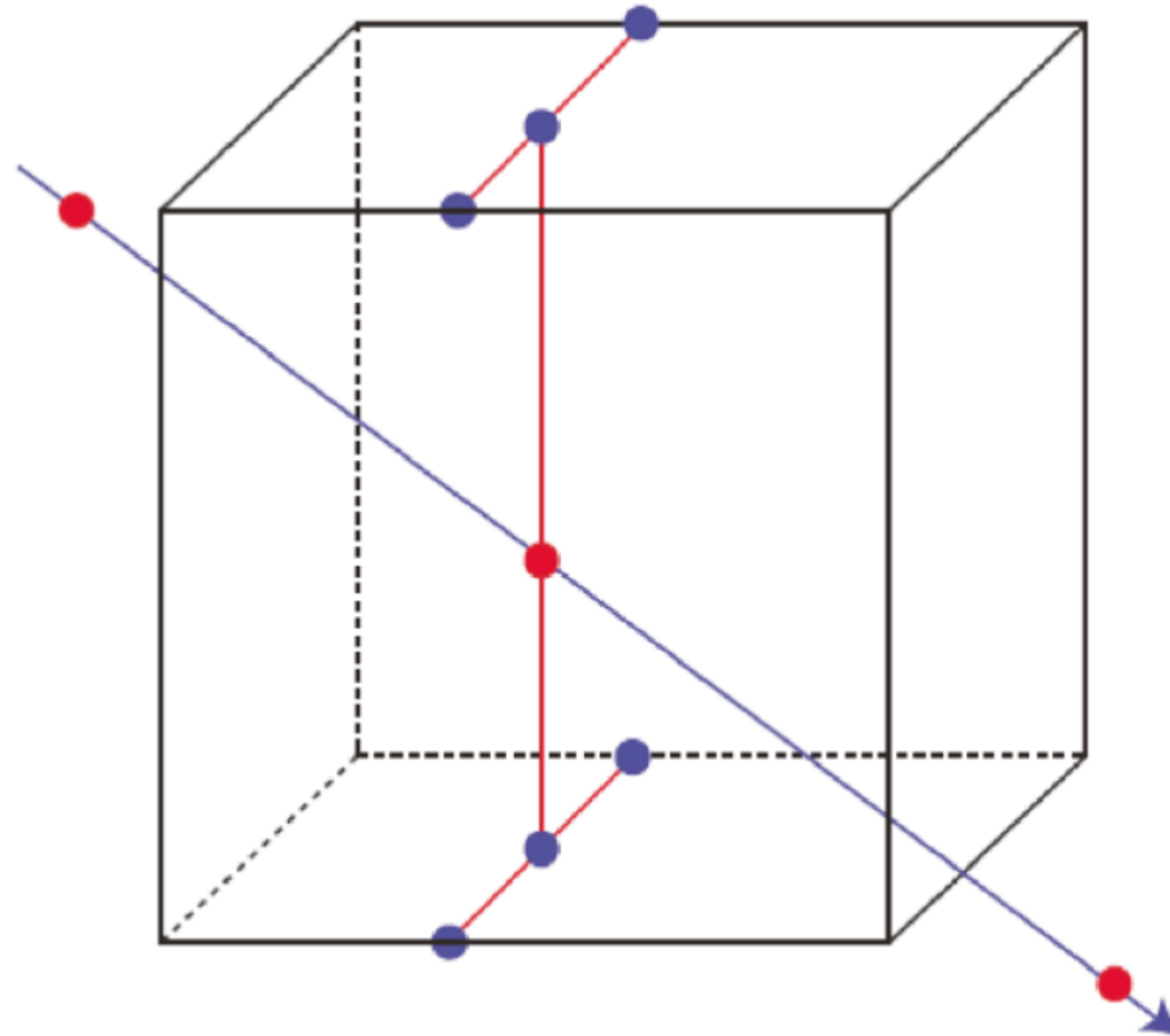
Bilinear Interpolation



$$f(x, y) = (1-v) ((1-u) f(x_i, y_j) + u f(x_{i+1}, y_j)) + v ((1-u) f(x_i, y_{j+1}) + u f(x_{i+1}, y_{j+1}))$$



Trilinear Interpolation



Gradient Computation

- **Gradients**

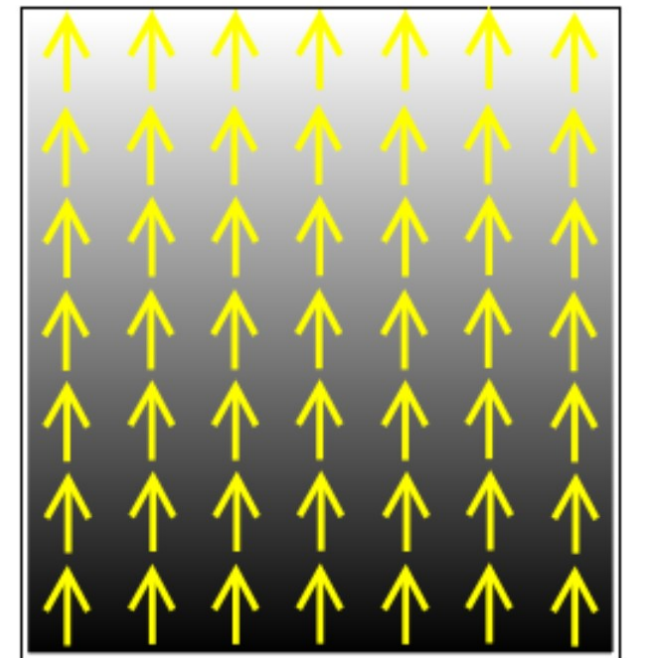
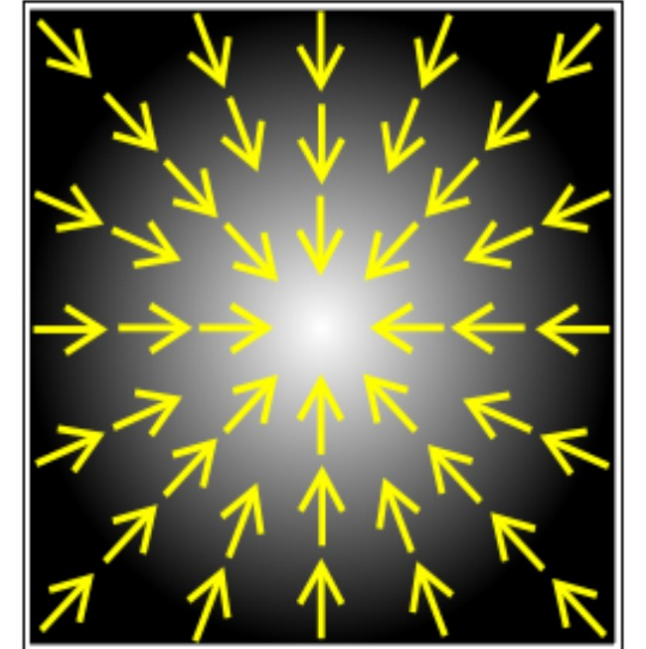
Magnitude and direction

Describe local changes in scalar field

- **Central difference**

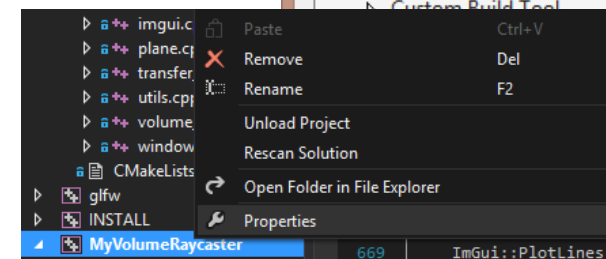
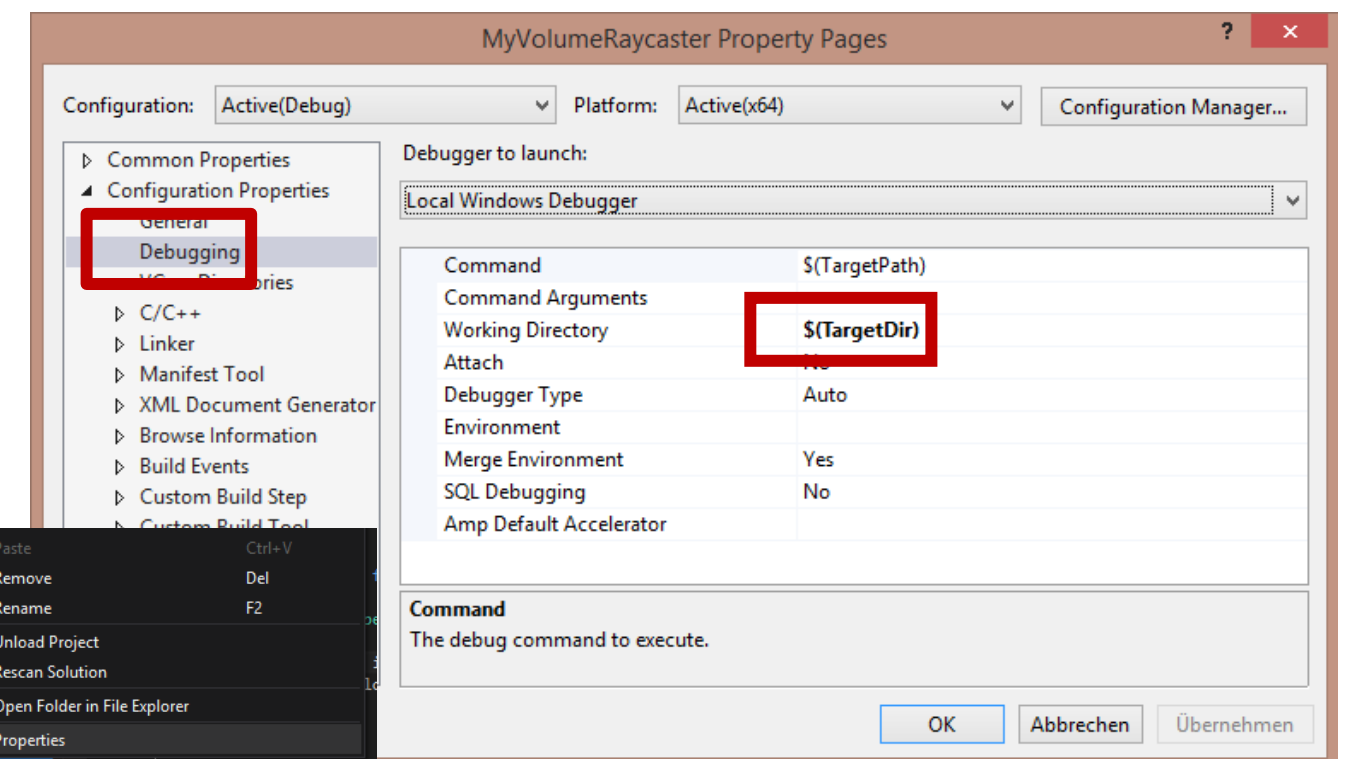
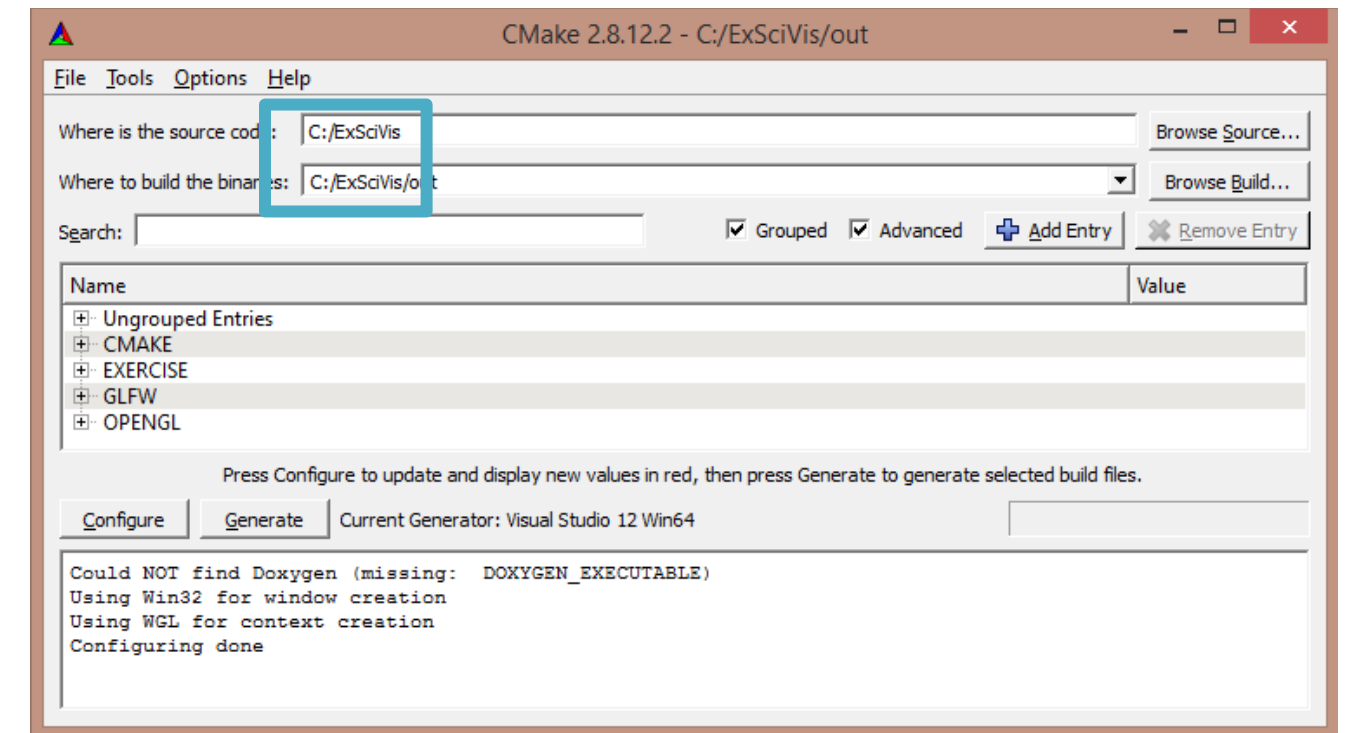
$$D_x = (f(x+1, y, z) - f(x-1, y, z)) / 2$$

Fast, easy to implement but only an approximation

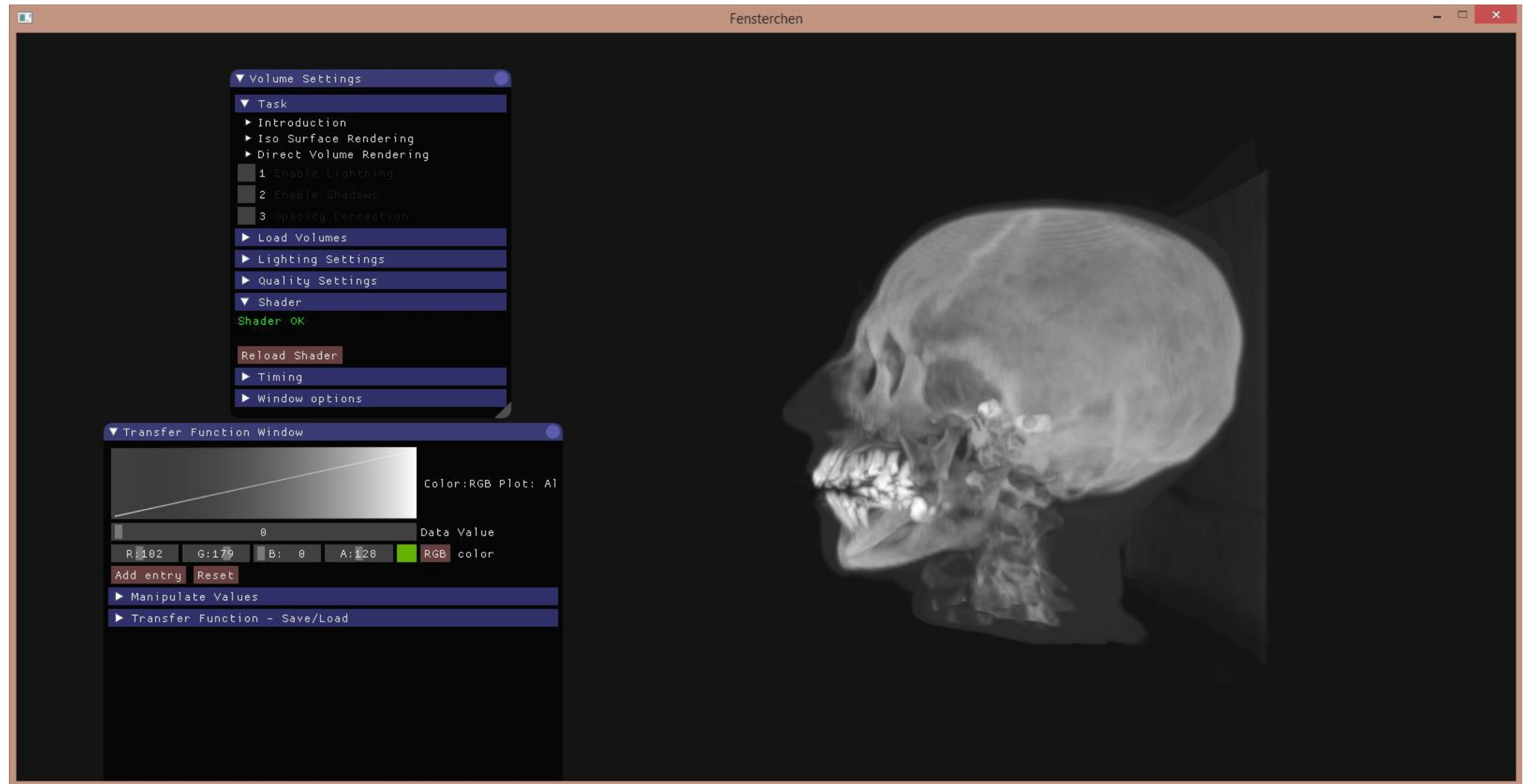


Project Setup

- Download .zip
<https://github.com/vrsys/ExSciVis2018/archive/master.zip>
- Or fork
<https://github.com/vrsys/ExSciVis2018.git>
- Generate Makefile with Cmake
 - Linux:** Cmake is installed, use cmake
 - Windows: download Cmake
<https://cmake.org/files/v3.11/cmake-3.11.2-win32-x86.zip>
- Build and run

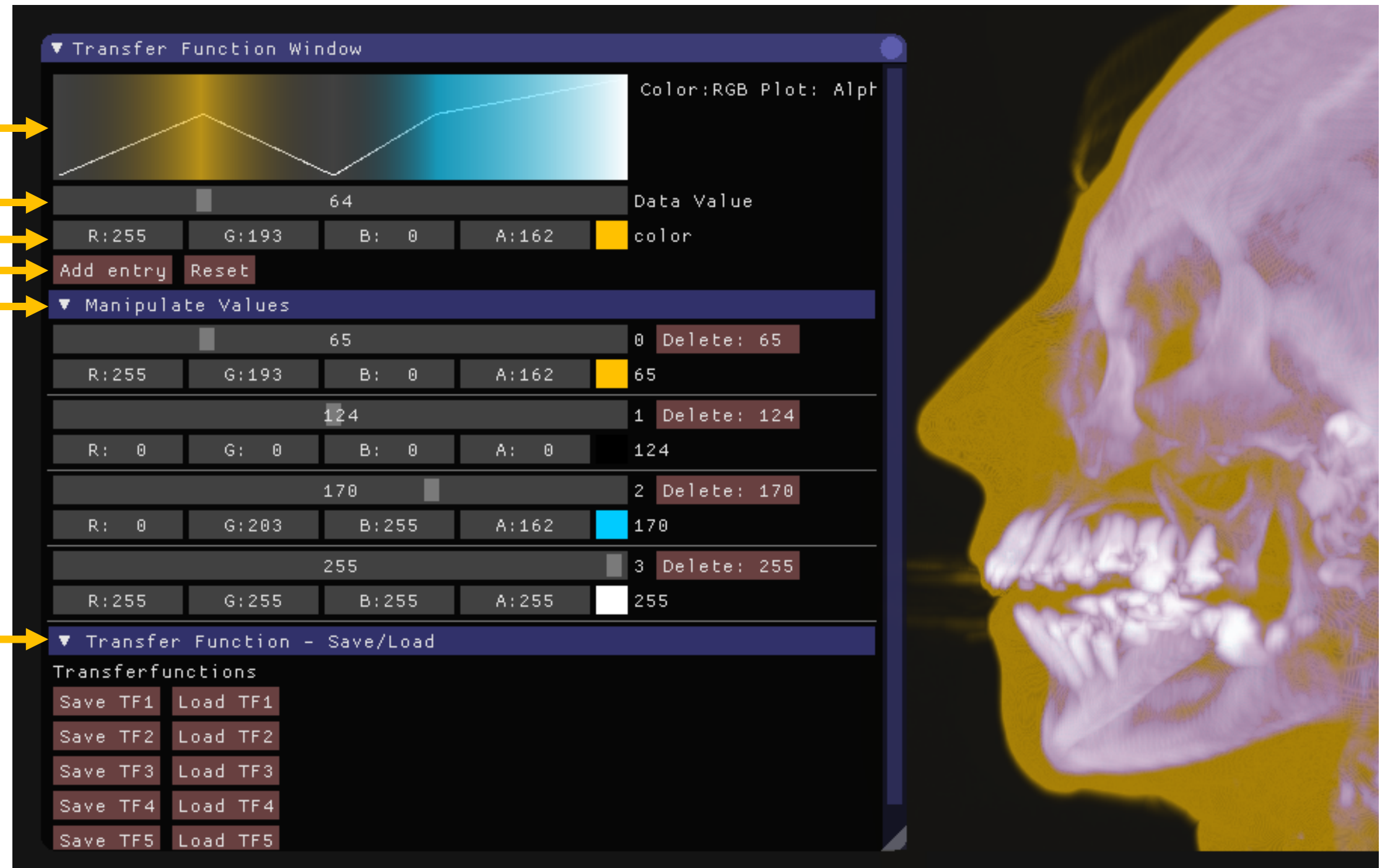


ExSciVis – First look



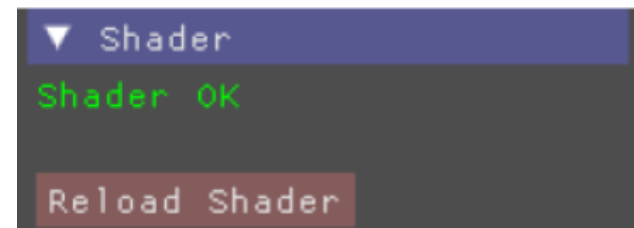
ExSciVis – Transfer Function Editor

- Current transfer function
- Set data value
- Set color / opacity
- Add stop
- Manipulate existing stops
- Save / Load



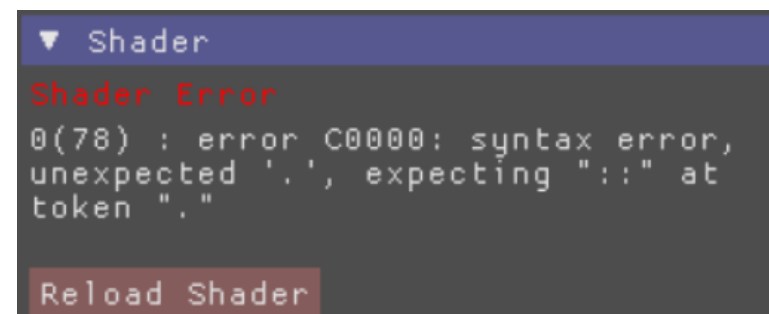
ExSciVis – GLSL

- Framework uses graphics card hardware acceleration
- Raycasting happens in **ExSciVis/source/shader/volume.frag**
- After editing this file, hit reload shader button



(no recompilation, no relaunch)

- If your code does not compile, you get a message



(program will keep running with last working shader)

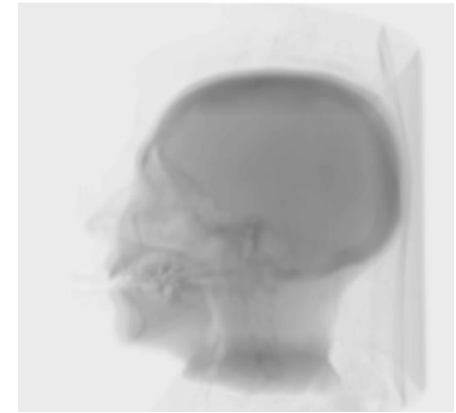
Maximum Intensity Projection

- Get data value at sampling point
- Evaluate transfer function
- Determine maximum color
- Move to next sampling position

```
66 #if TASK == 10
67     vec4 max_val = vec4(0.0, 0.0, 0.0, 0.0);
68     ....
69     // the traversal loop,
70     // termination when the sampling position is outside volume boundaries
71     // another termination condition for early ray termination is added
72     while (inside_volume)
73     {
74         // get sample
75         float s = get_sample_data(sampling_pos);
76         ....
77         // apply the transfer functions to retrieve color and opacity
78         vec4 color = texture(transfer_texture, vec2(s, s));
79         ....
80         // this is the example for maximum intensity projection
81         max_val.r = max(color.r, max_val.r);
82         max_val.g = max(color.g, max_val.g);
83         max_val.b = max(color.b, max_val.b);
84         max_val.a = max(color.a, max_val.a);
85         ....
86         // increment the ray sampling position
87         sampling_pos += ray_increment;
88
89         // update the loop termination condition
90         inside_volume = inside_volume_bounds(sampling_pos);
91     }
92
93     dst = max_val;
94 #endif
```

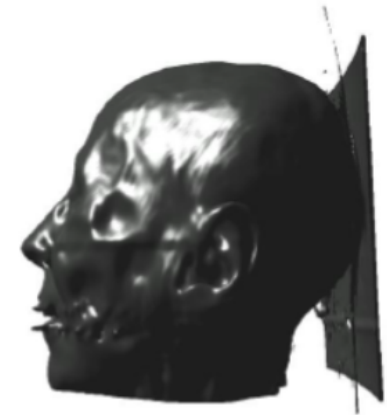

Assignment 1: Isosurfaces

1. Implement *average intensity projection*. (Hint: by default, the raycaster uses *maximum intensity projection*)
[1 Point]
2. Implement a *first-hit* ray traversal scheme for variable thresholds to visualize *isosurfaces*.
[1 Point]
3. Improve the intersection search using a *binary search* method. (Hint: do this after assignment 2, task 2 to see the difference)
[1 Point]



Assignment 2: Shading

1. Implement a function `get_gradient()` to calculate the gradient at a given volume sampling position. (Hint: central differences method suffices).
[1 Point]
2. Determine the *surface normal* for the found intersection point and calculate a basic illumination for the iso-surface. (Hint: Color-code and visualize your normals to make sure they are correct. A simple *phong shading model* suffices)
[1 Point]
3. Extend the illumination calculation for the correct display of surface shadows.
[1 Point]



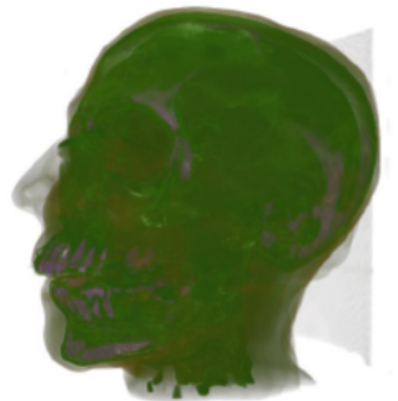
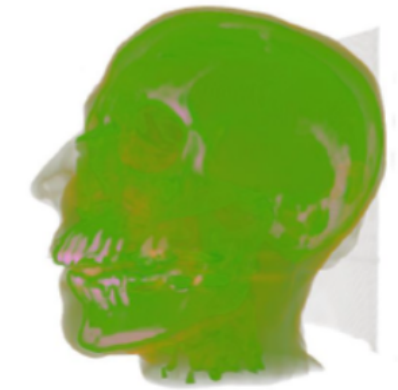
Assignment 3: Compositing

1. Implement a *front-to-back compositing* traversal scheme
[1 Point]

Implement a *back-to-front compositing* traversal scheme
[1 Point]

2. Use the generated volume gradients to calculate the local illumination for the volume samples during the compositing.
(Hint: simple *phong shading model* suffices)
[1 Point]

3. Extend the compositing algorithm with *opacity correction*.
[1 Point]



Optional Assignment

1. Based on the solution of assignment 3, implement *pre-integrated* volume rendering.
[optional, 4 Points]