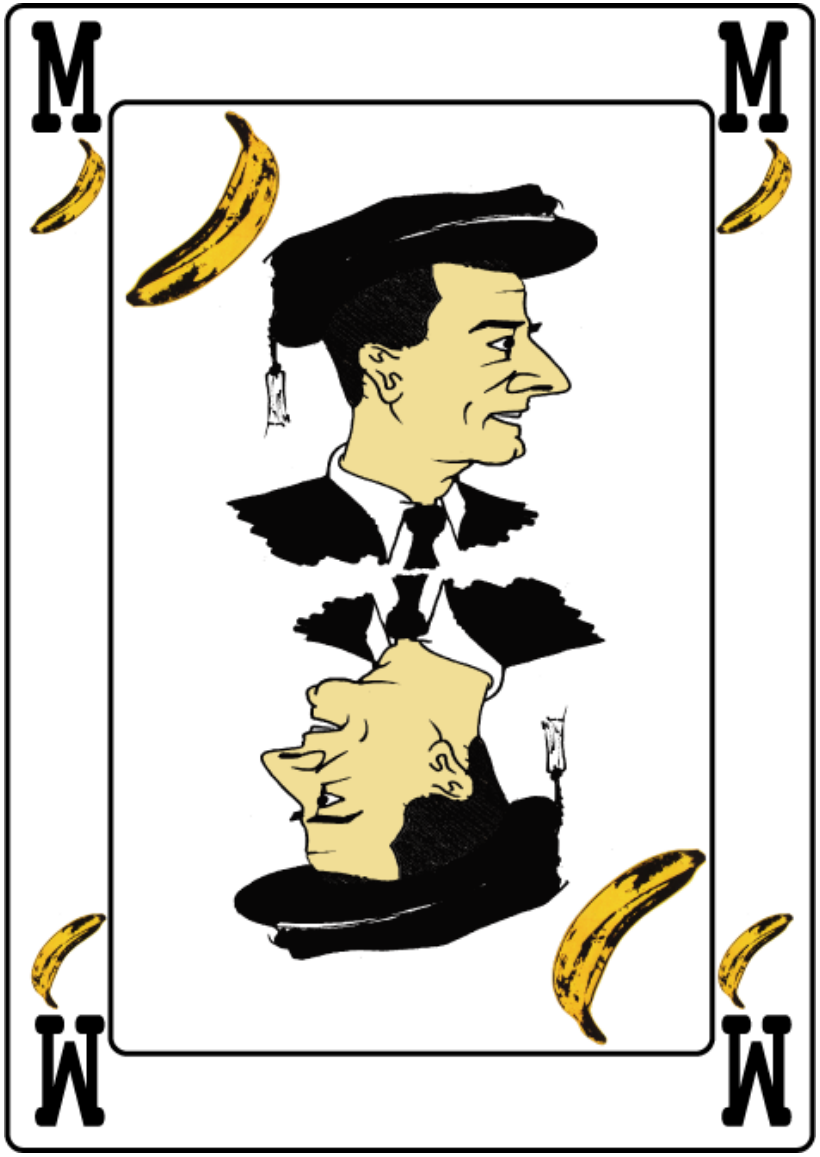
OpenGL & Visualization

Martin Ilčík

Institute of Computer Graphics and Algorithms

Vienna University of Technology

Motivation



- What is OpenGL
- How to use OpenGL
- Slices with OpenGL
- GPU raycasting

What is OpenGL?

- Low-level API for 3D graphics
  - Cross-platform
  - High extensible
  - Open source
  - OS independent
- Headers for all common prog. languages
- Lots of tutorials and resources on the web
- Very simple to use

What can it do?

- HW acceleration for
  - Rendering geometry
  - Texture lookups
  - Drawing buffers
- GPU programming
  - Programmable pipeline

What can't it do?

- 3D engine stuff
- GUI
- Math
- Window managing
- Input processing

What can we use it for? (1)

- Slices
  - Stored as texture(s)
  - Quick switching and manipulation
  - Arbitrary sliceplane
- Transfer function
  - Again a texture
  - Comfortable and easy changes
  - Fast application to data

What can we use it for? (2)

- Raycasting
  - with 2.0 shaders partially on GPU
  - with 3.0 shaders completely on GPU
  - for others at least pleasant drawing
- Other HW accelerated volume rendering techniques
  - discussed at lectures
  - also in EG'06 tutorial
  - not required for the LU

Let’s start!

Now I really want to use all the great benefits of OpenGL!

How?

OpenGL resources (1)

- http://www.opengl.org
  - News, Links, Forums (great response time)
  - OpenGL 1.x/2.x specs, GLSL specs
    - best help & reference
- The red book – old but good
- http://nehe.gamedev.net
  - Biggest and best tutorial series all over the net
  - NeHe auf deutsch
  - Featured articles on general programming

OpenGL resources (2)

- GLinfo (for win)
  - find out your graphics HW capabilities
- http://www.gamedev.net
  - Many resources, also for 3D graphics
  - Very good forums
- http://developer.nvidia.com, http://www.ati.com/developer
  - Many papers and presentations
  - Documentation for company specific features

Principles of OpenGL

- Structured, not object oriented
- State machine with a state-vector
- 1.x core is old, but regulary extended
  - ARB, EXT extensions
  - NV, ATI extensions
- 2.x common standard supported by HW
  - Functionality of 2.0 = Functionality of 1.5
- Function names start with **gl**
- Type of main parameter as suffix: **i, f, us, v …**

OpenGL setup (1)

- Application Initialization
  - Create the window
  - Attach the rendering surface to the OpenGL
  - Set it’s pixel format and buffers
- OpenGL initialization
  - Setup basic parameters and states
  - Extensions availability check
  - Window resizing handling
  - Load resources

OpenGL setup (2)

- Rendering loop
  - Clear the buffer
  - (Reset transformations)
  - Draw
- Setup can be done
  - Self-made
  - SDL
  - GLUT
- All samples and frameworks at
- http://nehe.gamedev.net/ … NeHe Basecode

Hello Triangle!

- Type this into the rendering loop

```
glBegin(GL_TRIANGLES);
glVertex3f(-1.0f, 0.0f, 0.0f);
glVertex3f(1.0f, 0.0f, 0.0f);
glVertex3f(0.0f, 1.0f, 0.0f);
glEnd();
```

- More in NeHe Lesson 2

Hello colorful triangle!

```
glBegin(GL_TRIANGLES);
glColor3f(1.0f, 0.0f, 0.0f);
glVertex3f(-1.0f, 0.0f, 0.0f);
glColor3f(0.0f, 1.0f, 0.0f);
glVertex3f(1.0f, 0.0f, 0.0f);
glColor3f(0.0f, 0.0f, 1.0f);
glVertex3f(0.0f, 1.0f, 0.0f);
glEnd();
```

- More in NeHe Lesson 3

Hello texture! (1)

- Enable the texturing (only 2^K textures)

  glEnable(GL_TEXTURE_2D);

- Allocate memory for texture data

  unsigned char* data = new unsigned char[size*size*3];

- Create alias for our texture

  GLuint texture;

- Genrate a new texture in the graphics memory

  glGenTextures(1, &texture);

Hello texture! (2)

- Send the texture data to the graphics card

```
glBindTexture(GL_TEXTURE_2D, texture);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB,
    size, size, 0, GL_RGB,
    GL_UNSIGNED_BYTE, data);
delete[] data;
```

- Set the bilinear interpolation (tent filter) !!!

```
glTextureParameteri(GL_TEXTURE_2D,
    GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTextureParameteri(GL_TEXTURE_2D,
    GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```


Hello textured quad!

```
//don’t forget to bind a texture before
glBegin(GL_QUADS);
glTexCoord2f(0.0f, 0.0f);
glVertex3f(-1.0f, -1.0f, 0.0f);
glTexCoord2f(1.0f, 0.0f);
glVertex3f(1.0f, -1.0f, 0.0f);
glTexCoord2f(1.0f, 1.0f);
glVertex3f(1.0f, 1.0f, 0.0f);
glTexCoord2f(0.0f, 1.0f);
glVertex3f(-1.0f, 1.0f, 0.0f);
```

- More in NeHe Lesson 6

Black hole?

- You'll see nothing, unless you fill the texture data with some values
- This has to be done before calling glTexImage
- Texture data
  - just a pointer to some memory block
  - copied to the graphics memory
- Try some constant color or just random noise
- Our volumetric data
  - is just a 3D image
  - even in raw format
- Related stuff in NeHe lesson 33

Slices as textures

- Generate separate texture for each slice for each axis
  - Statically precomputed (needs memory)
  - Dynamically generated on demand
- Save all the volume data in one 3D texture
  - Easy slicing for arbitrary plane
  - HW accelerated lookups (incl. interpolation)
  - Can be used as input for GPU raycasting
  - glTexImage3d, glTexCoord3f, GL_TEXTURE_3D

Reading the dataset

```
FILE* inData;
inData = fopen(filename, “rb”);
unsigned short dimx, dimy, dimz;
fread(&dimx, 1, 2, inData); //one entry
fread(&dimy, 1, 2, inData); //two bytes long
fread(&dimz, 1, 2, inData);
unsigned short* imData = new unsigned
    short[dimx*dimy*dimz];
fread(imData, dimx*dimy*dimz, 2, inData);
```


Slices as 2D textures (1)

```
ReadData(); //let the dataset be stored in data[]
unsingned shot* slicedata = new unsigned
    short[size*size];
k = 10;
for (int i = 0; i < dimx; i++)
  for (int j = 0; j < dimy; j++)
    slicedata[(size*j)+i] = data[(dimx*dimy*k) +
    (dimx*j) + i];
```


Slices as 2D textures (2)

```
GLuint slice; glGenTextures(1, &slice);
glBindTexture(GL_TEXTURE_2D, slice);
glTexImage2D(GL_TEXTURE_2D, 0,
    GL_LUMINANCE, dimx, dimy, 0, GL_LUMINANCE,
    GL_UNSIGNED_SHORT, slicedata);
//don't forget to set the filters
delete[] slicedata;
```

- for arbitrary data dimensions
  - GL_TEXTURE_RECTANGLE_ARB
  - or use gluBuild2DMipmaps
  - glTexSubImage2D

Slices as 3D textures (1)

```
ReadData();
unsingned shot* slicedata = new unsigned
    short[dimx*dimy*dimz];
//we convert the 12-bit values to 16-bit
for(int i = 0; i < dimx*dimy*dimz; i++)
    slicedata[i] = data[i]*16;
```

Slices as 3D textures (2)

```
GLuint slices; glGenTextures(1, &slices);
glBindTexture(GL_TEXTURE_3D, slice);
glTexImage3D(GL_TEXTURE_3D, 0,
    GL_LUMINANCE, dimx, dimy, dimz, 0,
    GL_LUMINANCE, GL_UNSIGNED_SHORT,
    slicedata);
//don't forget to set the filters
delete[] slicedata;
```

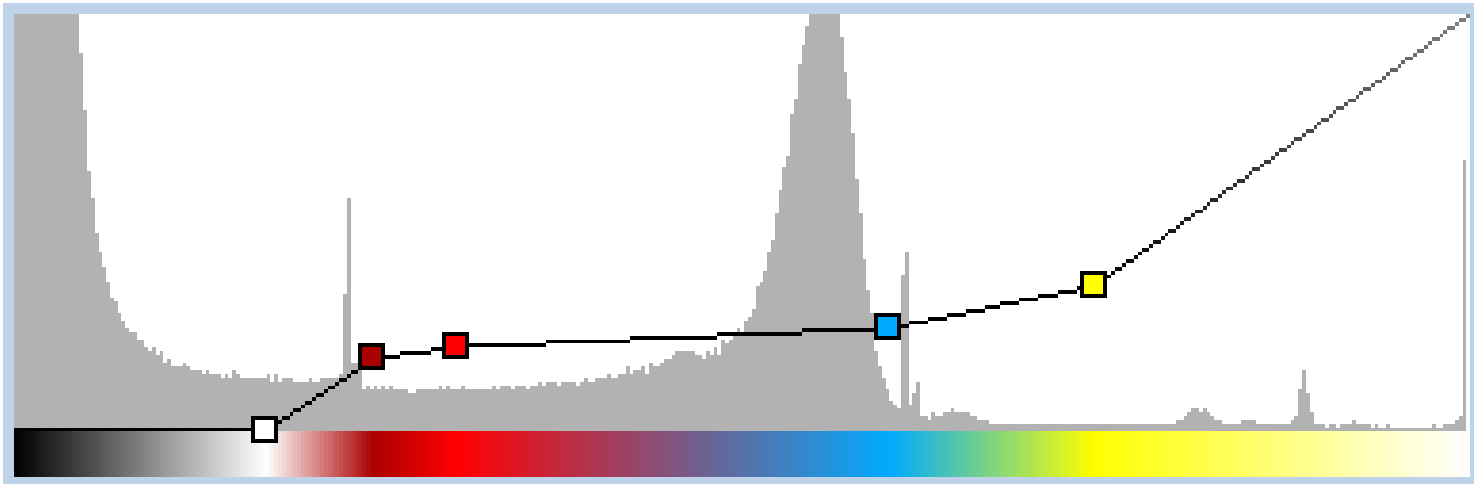
- for non 2^k data dimensions
  - GL_texture_non_power_of_two
  - or use gluBuild3DMipmaps
  - glTexSubImage3D

Slices as 3D textures (3)

```
k = 10.0f;
glBegin(GL_QUADS);
glTexCoord3f(0.0f, 0.0f, k/(float)dimz);
glVertex3f(-1.0f, -1.0f, 0.0f);
glTexCoord3f(1.0f, 0.0f, k/(float)dimz);
glVertex3f(1.0f, -1.0f, 0.0f);
//add the 2 remaining vertices in the same way
glEnd();
```


Transfer Function

- Interactive enhancement of the data by coloring
- [Intensity] → [RGBA]
  - Values defined at arbitrary points
  - Interpolation
  - What about trying to use L*a*b*



Transfer Texture

- a 1D texture
  - X axis intensity
  - [x]→[r,g,b,a]; $<0,1> \rightarrow <0,1>^4$
  - Easy to show to the user
  - Can be applied to slices/volumes in a shader
  - Automatic interpolation of colors
  - Must be regenerated after change
  - Sampling errors
    - Take a big texture (e.g. dimx == 4096)

Intermediate OpenGL

Resources

- Read more tutorials
- Download the OpenGL specs
- Get the latest headers (e.g. gl.h, glu.h, glext.h)
- Google
- Ask me :)

Transformations

- Matrices included in OpenGL state
  - Modelview glMatrixMode(GL_MODELVIEW);
  - Projection glMatrixMode(GL_PROJECTION);
- Reset the current matrix glLoadIdentity();
- Matrix stack
  - glPushMatrix();
  - glPopMatrix();
- Transformations
  - glTranslatef(…), glRotatef(…), glScalef(…)
  - glMultMatrix(…);
- More in this slides (Ed Angel, UNM)

Camera stuff

- Camera position and viewing direction
  - gluLookAt(eye,center,up);
- Projection type, clipping planes
  - glPerspective(angle, ratio, zNear, zFar);
  - glOrtho(left, right, bottom, up, zNear, zFar);
  - Should be updated by viewport changes
- More in this slides by Ed Angel, UNM

Memory management

- Objects allocated by OpenGL calls should be cleaned up, when no more needed
- Video RAM
  - Stores textures, shaders, displaylists, vertexarrays, VBO's, FBO's
  - When filled full, swapping to RAM – SLOW!
  - By visualization applications easily overflowed
- Use glDeleteTextures(…) etc.

Multitexturing

- glActivateTexture(GL_TEXTUREn);
  - Switches all texturing commands to the unit n
  - Now texture compositing modes come to play
    - glTexEnv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, mode)
    - GL_NONE, GL_REPLACE,GL_MODULATE
  - Deactivate texture units when no more used
    - glDisable(GL_TEXTURE_2D);
- glMultiTexCoord(GL_TEXTUREn, s, t …);
- More in OpenGL specs

Basic shaders stuff

- Low level GPU assembler (no more used)
- NVidia's Cg
  - Crossplatform (OpenGL, DirectX)
  - Different versions
- GLSL
  - Integrated in OpenGL 2.0
  - Compiled and linked at runtime
  - Works only with shaders 2.0 and higher
- CUDA

GLSL Shaders initialization

- Read source from a text file
- Create vertex/fragment shader object
  - glCreateShaderObjectARB(shader_type);
  - glShaderSourceARB(…);
  - glCompileShaderARB(shader);
- Create GPU program object
  - glCreateProgramObject();
  - glAttatchObjectARB(prog, shader);
  - glLinkProgramARB(prog);

Enabling GLSL shaders

- glUseProgramObjectARB(program);
- passing parameters to variables inside of the shader
  - glGetUniformLocationARB(program, name);
  - glUniformARB(location, value);
- sampler variables must be set this way
  - n = glGetUniformLocationARB(prog, sampler_name);
  - glUniform1iARB(n, texture_unit_number);
- More in this article at NeHe (by Florian Rudolf)

Transfer function in GLSL (1)

- Vertex shader just as fixed pipeline

```
void main()
{
    gl_TexCoord[0] = glMultiTexCoord0;
    gl_TexCoord[1] = glMultiTexCoord1;
    gl_Position = gl_ModelViewProjectionMatrix *
      gl_Vertex;
}
```


Transfer function in GLSL (2)

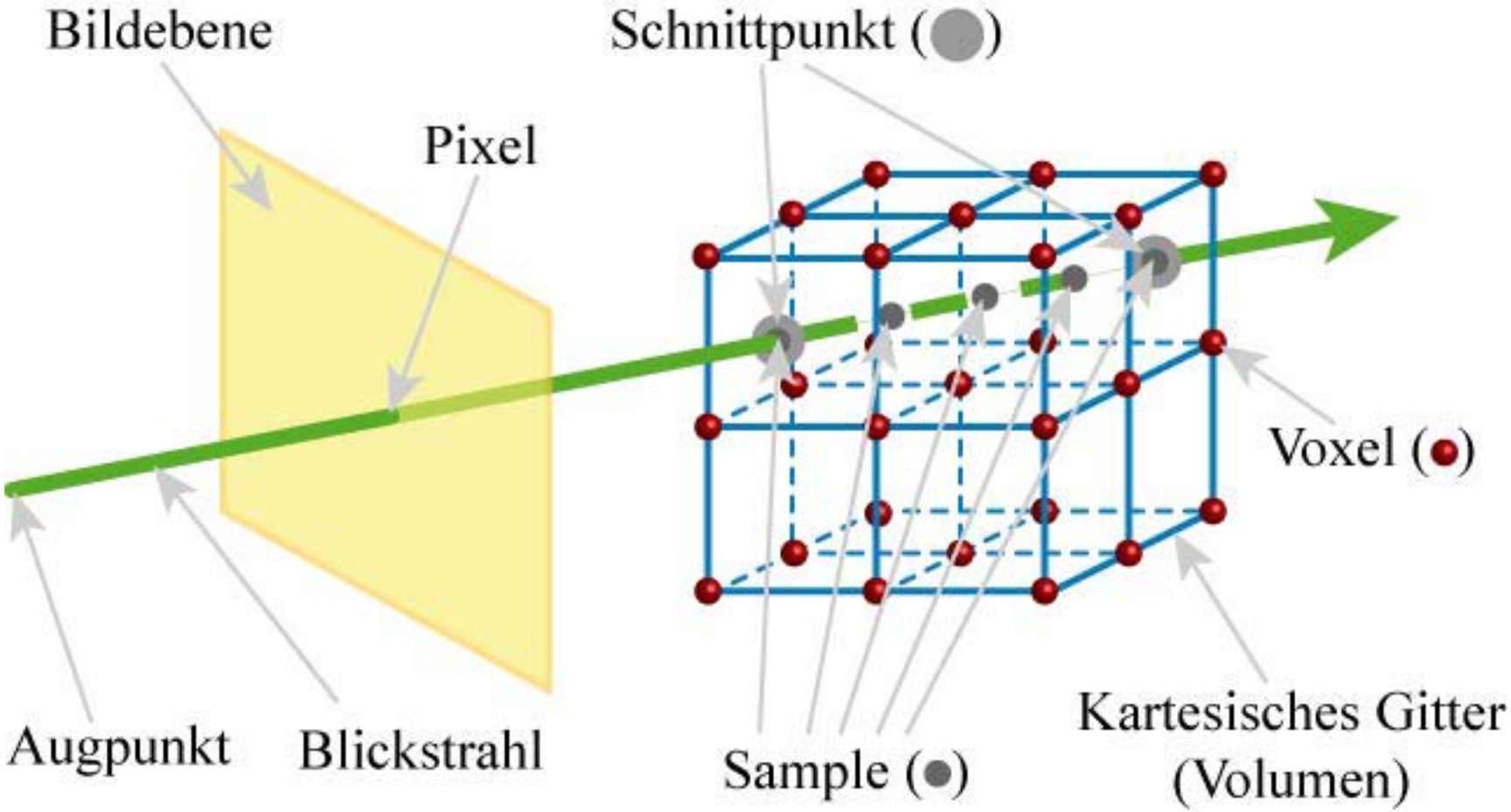
- Fragment shader

```
uniform sampler2D slice_sampler;
uniform sampler1D transfer_sampler;
void main()
{
    vec4 intensity = texture2D(slice_sampler,
      vec2(gl_TexCoord[0]) );
    gl_FragColor = texture1D(transfer_sampler,
      intensity.x);
}
```


GPU Raycasting

Advanced OpenGL

How it works (1)



How it works (2)

- Implementation
  - One fragment (pixel) = one ray
  - HW makes it simpler
- Implementation in shaders
  - Use direction textures for the ray
  - Iterate all the steps in a fragment shader
- Data storage
  - Render directions to textures
  - Store data in one or more 3D textures
  - Store the transfer function also in a texture

Direction textures

- Lowpoly envelope around the dataset
  - Take simply a box ☺
- Map world coordinates to it’s primary color
- Render
  - Front faces for entry points of rays
  - Back faces for exit points of rays
  - Store in two textures

Hardware ray initialization

- Rendering in 3 passes
  - 1st ... get texture with ray-entry coordinates
  - 2nd ... get texture with ray-exit coordinates
  - 3st ... actual raycasting
- In the fragment shader
  - Compute ray directions
  - Sample trough the dataset
    - iterate steps until opacity ~ 1.0 or ray outside

Rendering to texture

- Frame buffer object
  - New extension in OpenGL 1.5
  - Finally fast access to offscreen buffers
  - API feature, independent from hardware
  - Backwards compatible, depends on drivers
  - Interface similar to multitexturing
- More on LWJGL Wiki

Creating texture storage

```
glGenTextures(1, FBOTextureFront);
glBindTexture(
    GL_TEXTURE_RECTANGLE_ARB,
    FBOTextureFront);
glTexImage2D(
    GL_TEXTURE_RECTANGLE_ARB, 0,
    GL_RGBA8, ScreenX, ScreenY, 0,
    GL_RGBA, GL_UNSIGNED_BYTE, 0);
```


Creating and linking a FBO

```
glGenFramebuffersEXT(1, &FBufferFront);
glBindFramebufferEXT(
    GL_FRAMEBUFFER_EXT, FBufferFront);
glClearColor(0, 0, 0, 0);
glFramebufferTexture2DEXT(
    GL_FRAMEBUFFER_EXT,
    GL_COLOR_ATTACHMENT0_EXT,
    GL_TEXTURE_RECTANGLE_ARB,
    FBOTextureFront, 0);
```

- Create a FBO for the back faces just the same way

Rendering direction textures (1)

- Bind the front faces buffer and draw them

```
glBindFramebufferEXT(
    GL_FRAMEBUFFER_EXT, FBufferFront);
glClear(GL_COLOR_BUFFER_BIT);
glUseProgramObjectARB(Prog1st);
drawBoundingBox();
```


Rendering direction textures (2)

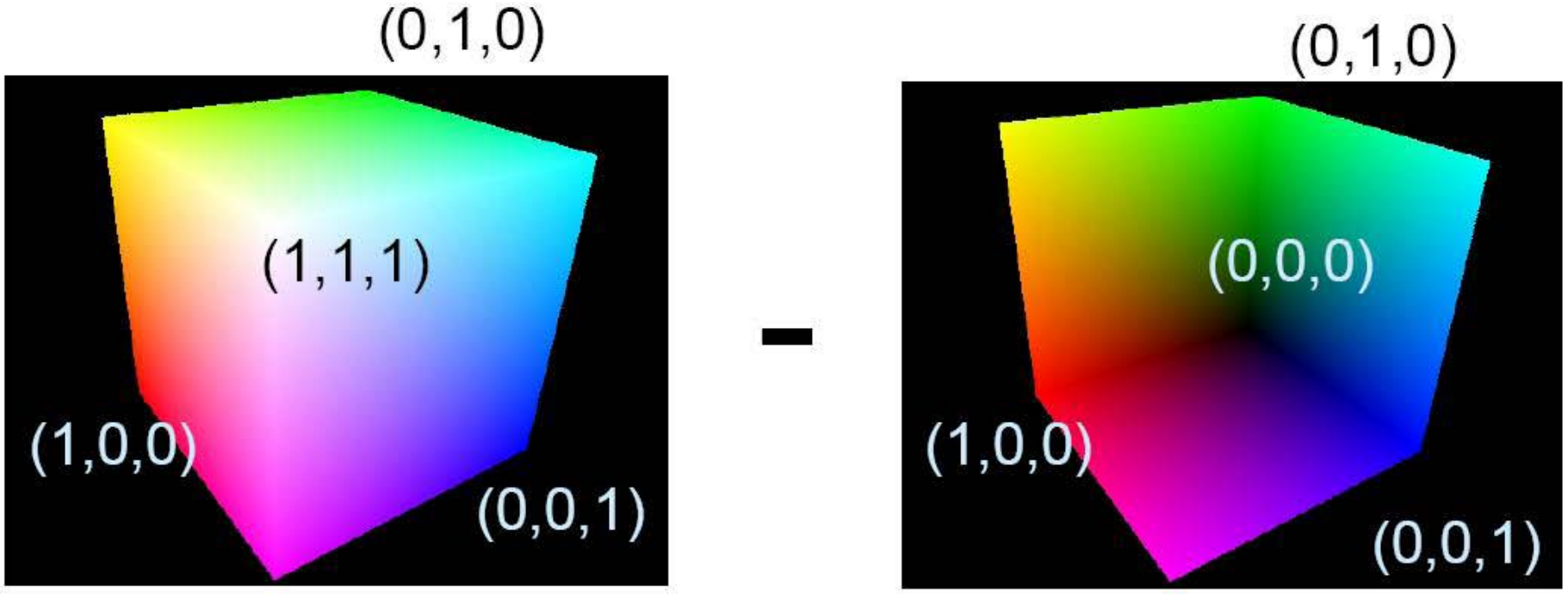
- Bind the back faces and draw them

```
glCullFace(GL_FRONT);
glBindFramebufferEXT(
    GL_FRAMEBUFFER_EXT, FBufferBack);
glClear(GL_COLOR_BUFFER_BIT);
drawBoundingBox();
```

- unbind the texture and render to screen

```
glBindFramebufferEXT(
    GL_FRAMEBUFFER_EXT, 0);
glCullFace(GL_BACK);
```

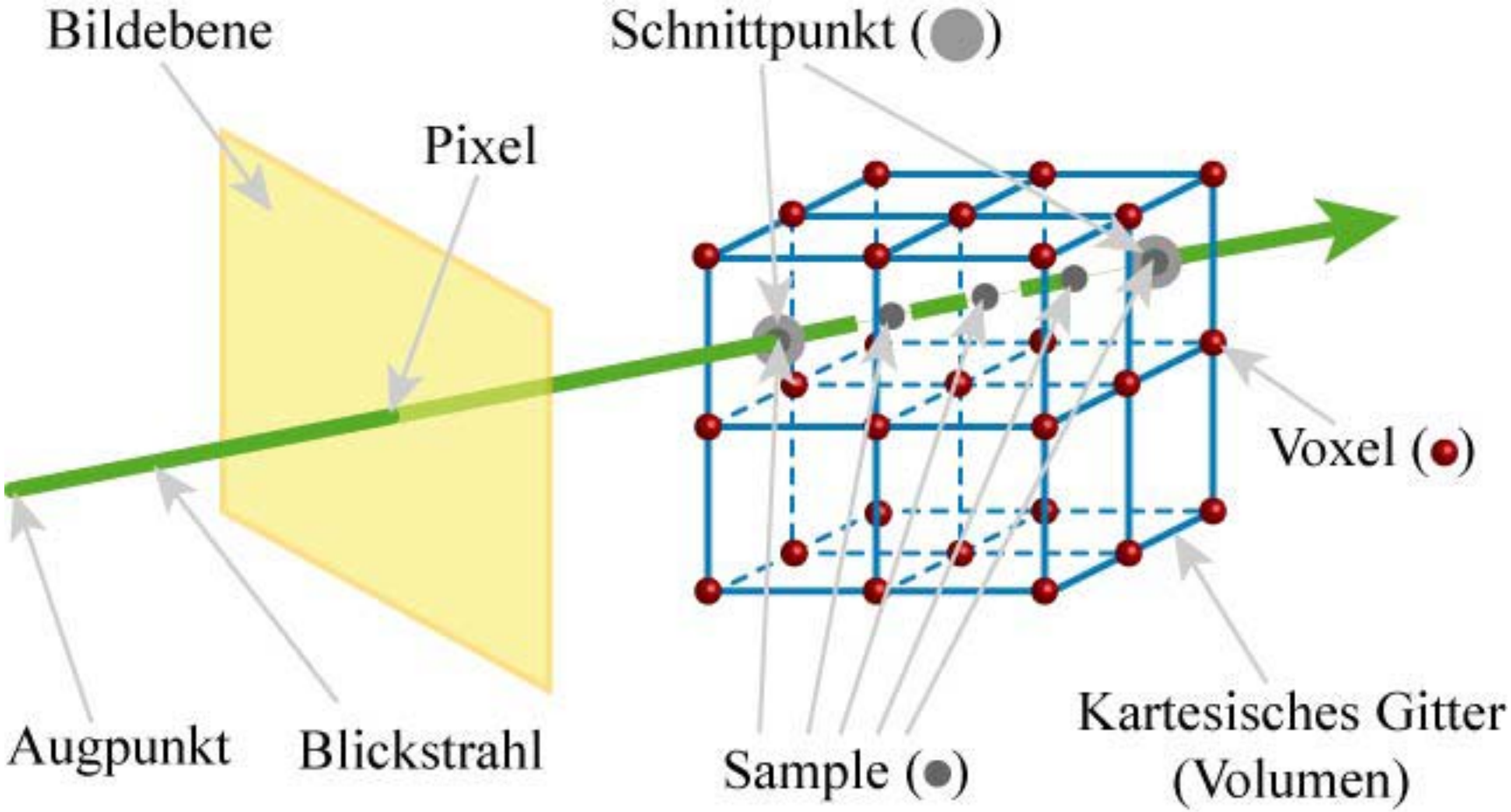

Rendering direction textures (3)



Raycasting in fragment shader

- Check if the ray crosses the bounding box
- Read the entry and exit point from textures
- Determine ray direction
- Iterate until opacity ~ 1.0 or outside
  - Read the intensity at this sample
  - Apply transfer function
  - Compositing
  - Next sample

How it works once again



Fragment shader (1)

- Input samplers

```
uniform sampler2DRect tex_entry;
uniform sampler2DRect tex_exit;
uniform sampler3D tex_intensity;
uniform sampler1D tex_transfer;
```


Fragment shader (2)

```
void main()
{
    const float dz = 0.005;
    const int maxrange = 347;
    vec4 entry_point = texture2DRect(tex_entry,
      vec2(glTexCoord[0]));
    vec4 exit_point = texture2DRect(tex_exit,
      vec2(glTexCoord[0]));

    float dist = distance(entry_point, exit_point)/dz;
    int maxiter = int(floor(dist));
    vec3 diff = (exit_point.xyz – entry_point.xyz)/dist;
```


Fragment shader (3)

```
if (entry_point.w == 0.0) discard;
else
{
    float numSamples = 0.0f;
    for(int = 0; i < maxrange; i++)
    {
       // see next slide
    }
    glFragColor = Result;
}
```


Fragment shader Compositing (4)

```
intensity = texture3D(tex_intensity, point);
transferred = texture1D(tex_transfer, intensity.x);

Result.xyz += (1.0 – Result.w) * transferred.w *
    transferred.xyz;
Result.w += (1.0 – Result.w) * transferred.w;

point +=diff;
if ((Result.w >= 1.0)||(i >= maxiter)) break;
```


Fragment shader Averaging (5)

```
intensity = texture3D(tex_intensity, point);
transferred = texture1D(tex_transfer, intensity.x);

Result.xyz += transferred.w * transferred.xyz;
Result.w += transferred.w;

point +=diff;
if (i >= maxiter)
{
    Result /= float(maxiter);
    break;
}
```


Conclusions

- Use hardware rendering
- Use the best API for you
- Store data in textures
- Shaders make things much simpler and faster

- Feel free to ask: ilcik at cg.tuwien.ac.at
- Thank you for your attention!