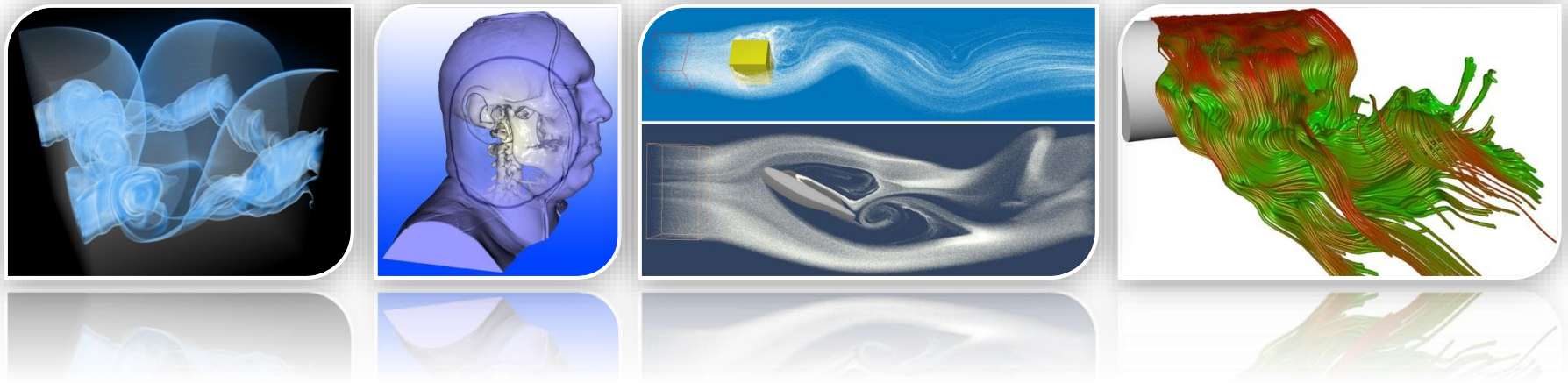
Master Practical Course

Interactive Visual Data Analysis



tum.3D

computer graphics & visualization

Today

- Assignment 5
  - (Volume-) Sequences
  - Volume Rendering Cleanup

Sequence Data

- `.dat` contains additional metadata:

```
ObjectFileName: pmlow_t%03iz.raw
Resolution:     128 128 128
Format:         BYTE
SliceThickness: 1 1 1
ObjectIndices:  0 60 4
```

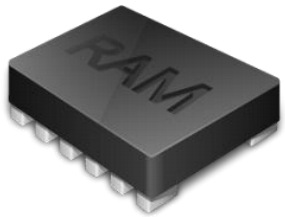
`.raw` file name pattern

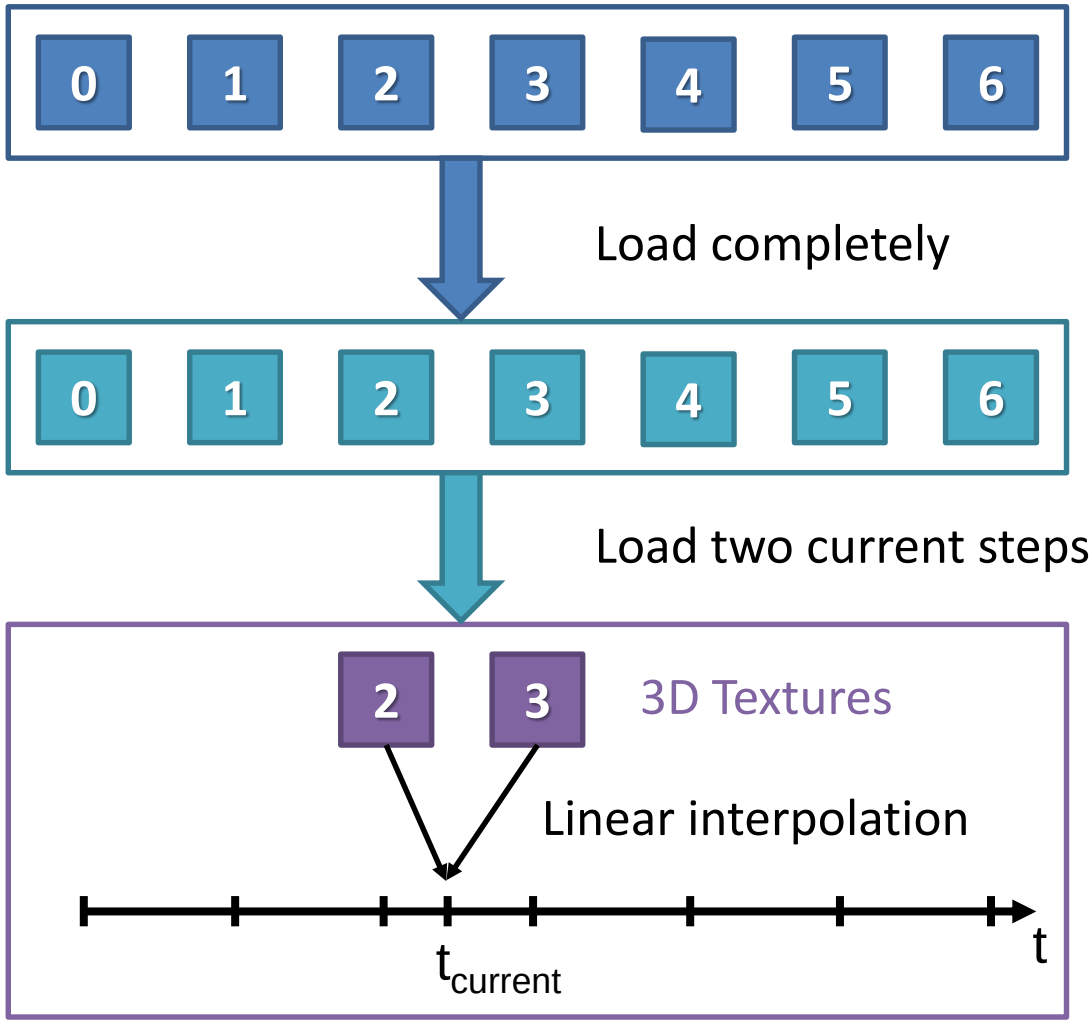
File indices: Min, Max and Increment

- Pattern in „printf“ syntax
  - Use `sprintf_s()` to generate filenames

Memory Strategy

- Assumption: Sequence fits completely into CPU RAM



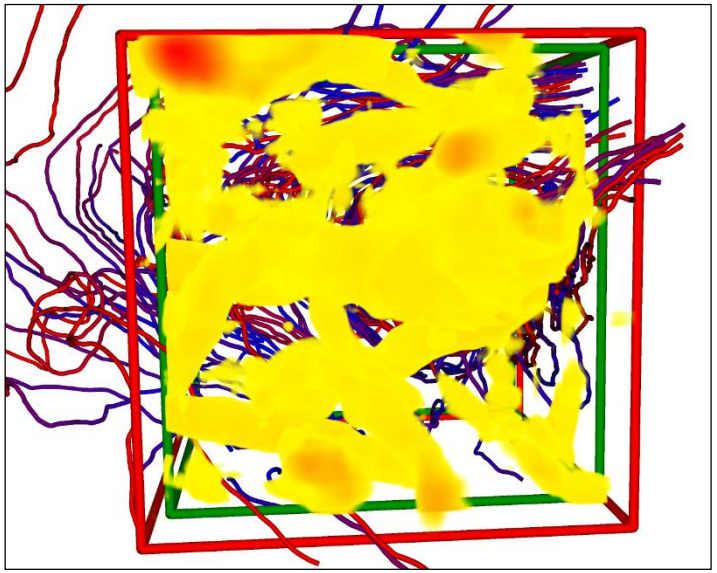


3D Texture Management

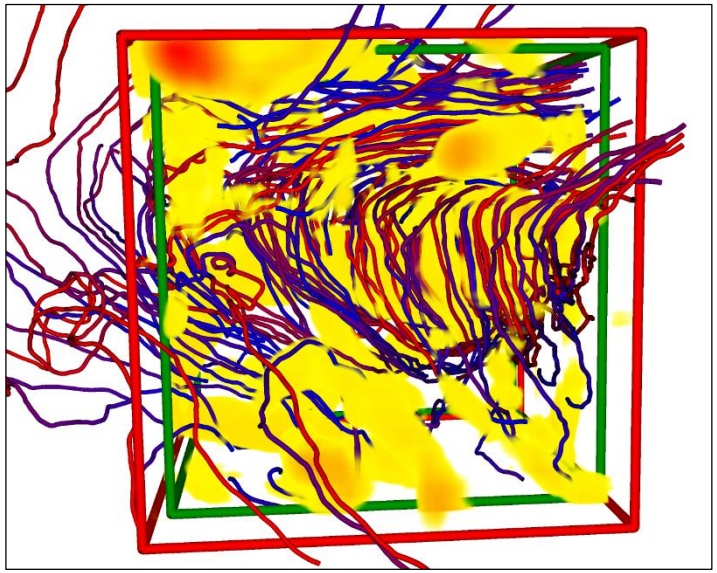
- Two 3D texture objects on GPU
  - Replace each existing `sample` with two `samples` + linear interpolation
  - Do **not** `Release` and Re-`Create` texture objects whenever the data changes!
  - Instead `UpdateSubresource()` (= CPU to GPU memory copy)

Volume Rendering and z-Test

- Render at the same time
  - Opaque geometry (particles, meshes, …)
  - Volume (DVR or iso-surface)
- Problem: Skip occluded parts of volume data



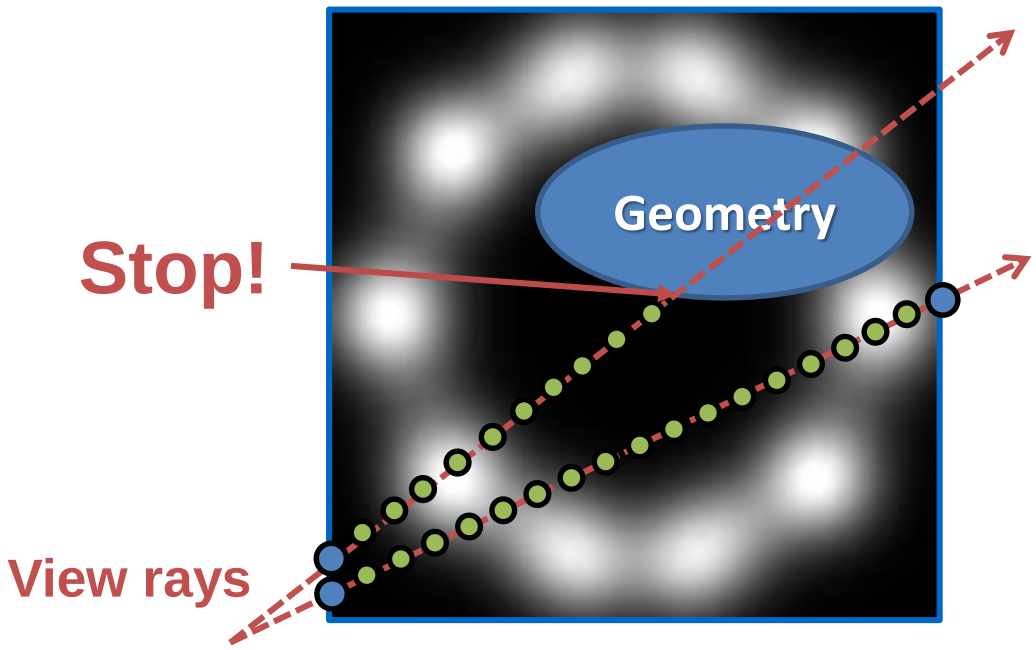
Wrong



Correct

Volume Rendering with Depth Buffer

- Solution:
  - Read depth buffer in pixel shader
  - Stop ray marching when $\text{depth}_{\text{ray}} > \text{depth}_{\text{depthBuffer}}$



Reading Depth Buffer

- Tell DXUT to use a readable depth buffer format (default is `DXGI_FORMAT_D24_UNORM_S8_UINT`)

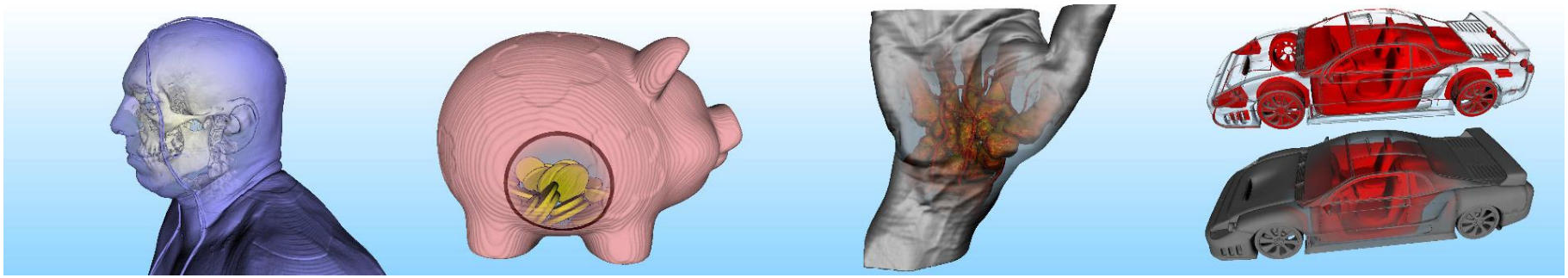
```
bool ModifyDeviceSettings(DXUTDeviceSettings* pDeviceSettings, …)  {
    pDeviceSettings->d3d11.AutoDepthStencilFormat = DXGI_FORMAT_D32_FLOAT;
    …
}
```

- Create a `DXGI_FORMAT_R32_FLOAT` texture with screen resolution to mirror depth buffer
  - Hint: `OnD3D11ResizedSwapChain() / OnD3D11ReleasingSwapChain()`

Reading Depth Buffer

- Immediately before drawing volume
  - Copy current depth buffer to this texture…
    - Get the depth buffer texture resource with `DXUTGetD3D11DepthStencilView()->GetResource()`
    - Copy texture with `ID3D11DeviceContext::CopyResource()`
  - …and bind it as shader resource
- Convert depth
  - Depth buffer stores depth $\in [0; 1]$ (NDC)
  - Use M_{Proj}^{-1} to get a view space depth (don‘t forget dehomogenization!)
- *Note: Direct access to the depth buffer would be more efficient, but is slightly more complicated*

ClearView (optional)



- Focus & Context technique
  - Outer isosurface: context
  - Inner isosurface: focus
  - Focus region: opacity kernel
    - based on 3D distance to point on surface
- Requires picking ☹
  - Read 3D mouse pointer position from depth buffer

wwwcg.in.tum.de/research/research/publications/2006/clearview-an-interactive-context-preserving-hotspot-visualization-technique.html

Questions ?