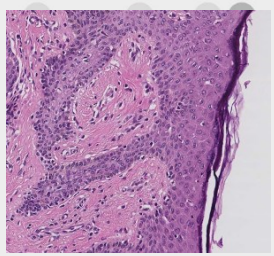
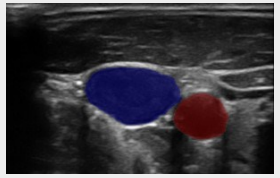
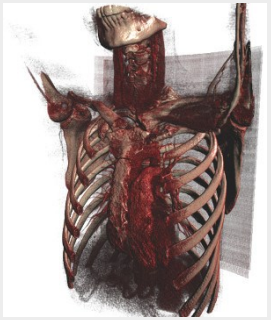






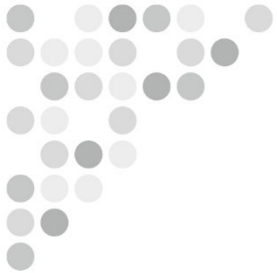
FAST
Framework for Heterogeneous Medical Image Computing and Visualization

OpenCL™

FAST: A Framework for High-Performance Medical Image Computing and Visualization

Erik Smistad
SINTEF Medical Technology &
Norwegian University of Science and Technology (NTNU)

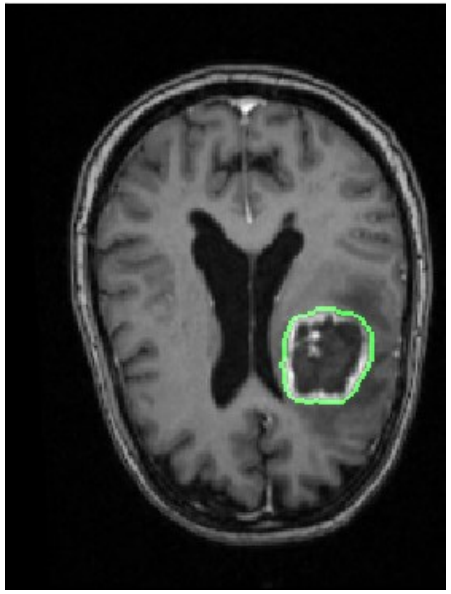
https://github.com/smistad/FAST

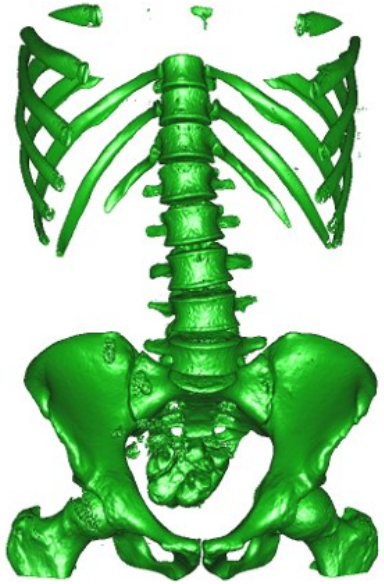


Medical imaging needs high-performance computing & visualization

CT & MRI

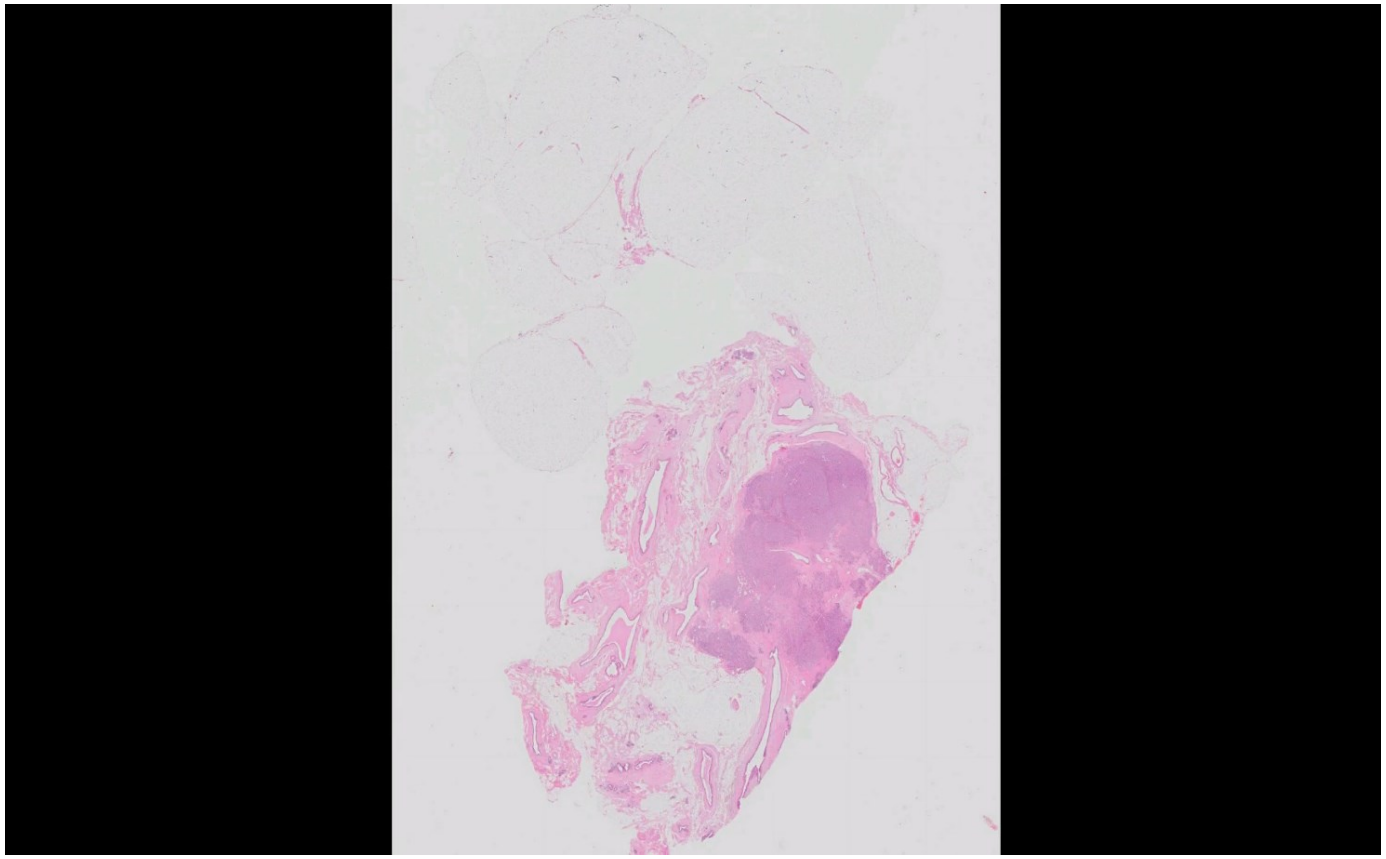
- 3D image data
- Large images
- 512 x 512 x 700 pixels





Digital pathology

- Digitized microscopy images of tissue samples
- Huge images
  - 40x magnification
  - 200,000 x 100,000 pixels
  - ~50 GB uncompressed
- Can’t fit into RAM
  - Virtual memory
  - Tiled image pyramids



Video courtesy of André Pedersen and NTNU IKOM

Ultrasound

- Real-time
- Compact devices
- 3D ultrasound

Screen capture of the
automatic measurement system

2013 - Goals

- Create a framework which makes high-performance computing and visualization of medical images:
  - Easy – Less code
  - Efficient – Use dedicated GPU, integrated GPU & multi-core CPU
  - Open-source
  - Cross-platform
    - Operating system (Windows, Linux, (Mac))
    - CPU/GPU vendor (AMD, Intel, NVIDIA)

How?

- In 2012-2014:
  - Very little GPU image processing support in existing frameworks
    - OpenCV
    - VTK, ITK
  - Ad-hoc
  - Not easy-to-use
- Our solution:
  - Create a new framework from scratch
  - GPU processing is the default
  - OpenCL (chosen over CUDA since not cross-vendor)
  - OpenGL (chosen over DirectX since not cross-OS)

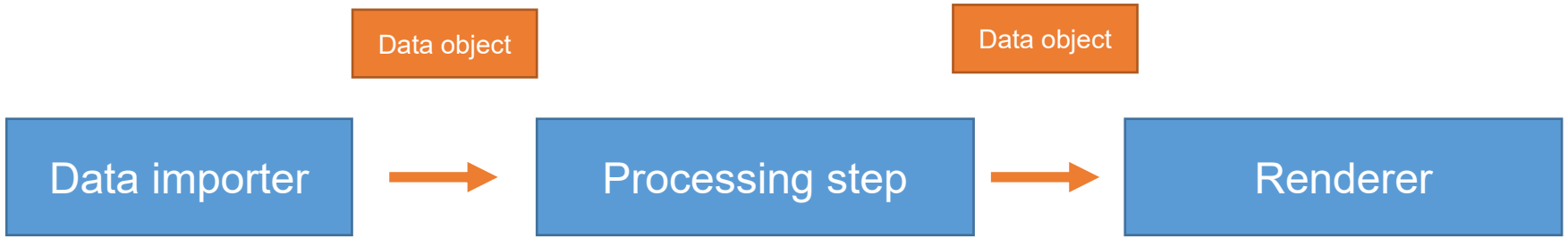
Processing pipeline concept

- Define pipeline first, then execute. Concept from VTK
- Demand driven execution pipeline
  - Will only re-execute if needed (parameter change, new input)
- Loose coupling
- Interchangeable process objects

```
auto importer = ImageFileImporter::New();
importer->setFilename("path/to/some_image.jpg");

auto processing = ProcessingStep::New();
processing->setSomeParameter(value);
processing->setInputConnection(importer->getOutputPort());

auto renderer = Renderer::New();
renderer->setInputConnection(processing->getOutputPort());
```

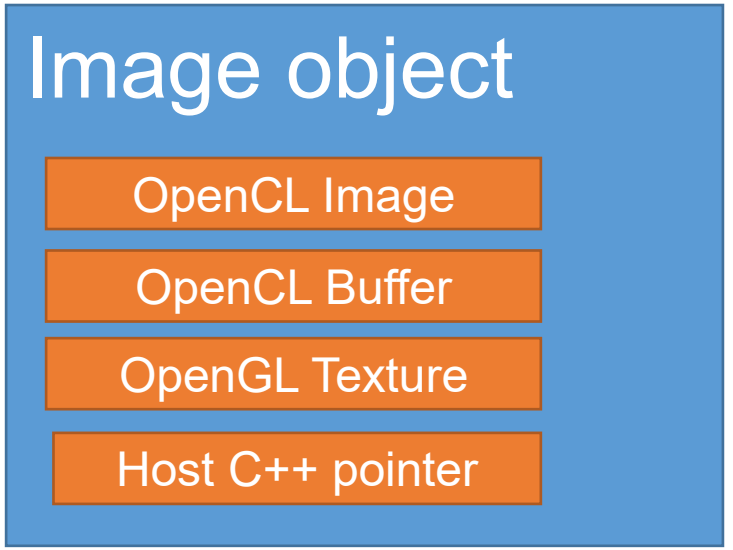


How FAST uses OpenCL

- Each process object has access to OpenCL devices
  - Can execute kernels
  - Can request access to data on OpenCL devices
- Reduce data transfers between CPU-GPU
- Most algorithms are implemented in OpenCL for GPUs
  - Surface extraction / Marching Cubes
  - Non Local Means Filtering
  - Convolutions – Gaussian, LoG
  - Morphology
  - Centerline extraction + Skeletonization
  - Gradient vector flow
  - Resampling/resizing/cropping/slicing/patch generation
  - Segmentation: Seeded region growing, thresholding, leve sets
  - Block/Template matching
  - Image and volume rendering
  - Surface rendering
  - ++++

How FAST uses OpenCL

- Data objects
  - The actual data can be stored in multiple locations
    - Image
      - OpenCL Image
      - OpenCL Buffer
      - Host C++ pointer
      - OpenGL Texture
    - Mesh (Vertices, Lines, Triangles)
      - OpenCL Buffer
      - OpenGL VBO, EBO
      - Host C++ std::vector
  - FAST keeps data coherent across all locations
  - Lazy data coherency – Request access



The early years < 2014

- A lot of frustrating bugs in OpenCL implementations
  - On Apple OpenCL: Kernel causing crash for no apparent reason. Changed order of code lines.
  - On AMD OpenCL: Machines with regional settings where comma as decimal point causing failure to compile OpenCL kernels.

```
// There is a bug in AMD OpenCL related to comma (,) as decimal point
// This will change decimal point to dot (.)
struct lconv * lc;
lc = localeconv();
if(strcmp(lc->decimal_point, ",") == 0) {
    setlocale(LC_NUMERIC, "C");
```

```
        // OpenCL on Mac is missing the mix function for some reason
#ifdef MAC_HACK
        const float3 point0f = (float3)(point0.x, point0.y, point0.z);
        const float3 point1f = (float3)(point1.x, point1.y, point1.z);
        const float3 vertex = (point0f + (point1f-point0f)*diff)*spacing;
```

OpenCL in FAST 2021

- OpenCL has matured
- Still using OpenCL version 1.2
- For every extension used, alternatives has to be implemented and maintained if not supported by all platforms.
- Gave up Mac OS X support when CL/GL was deprecated
  - Running OpenCL on Metal?

The big challenge

Cross-platform support
Low code complexity
Generality

Optimal Performance

Medical image data in FAST

- Most medical images are monochrome (1 channel only)
- Varying precision
  - Ultrasound – 2D uint8
  - CT – 3D int16
  - MR – 3D uint16
  - Microscopy – 2D RGB uint8
- FAST should handle all these image types
  - Every process object should handle different data types
- OpenCL Image data type is used extensively

Medical image data in FAST

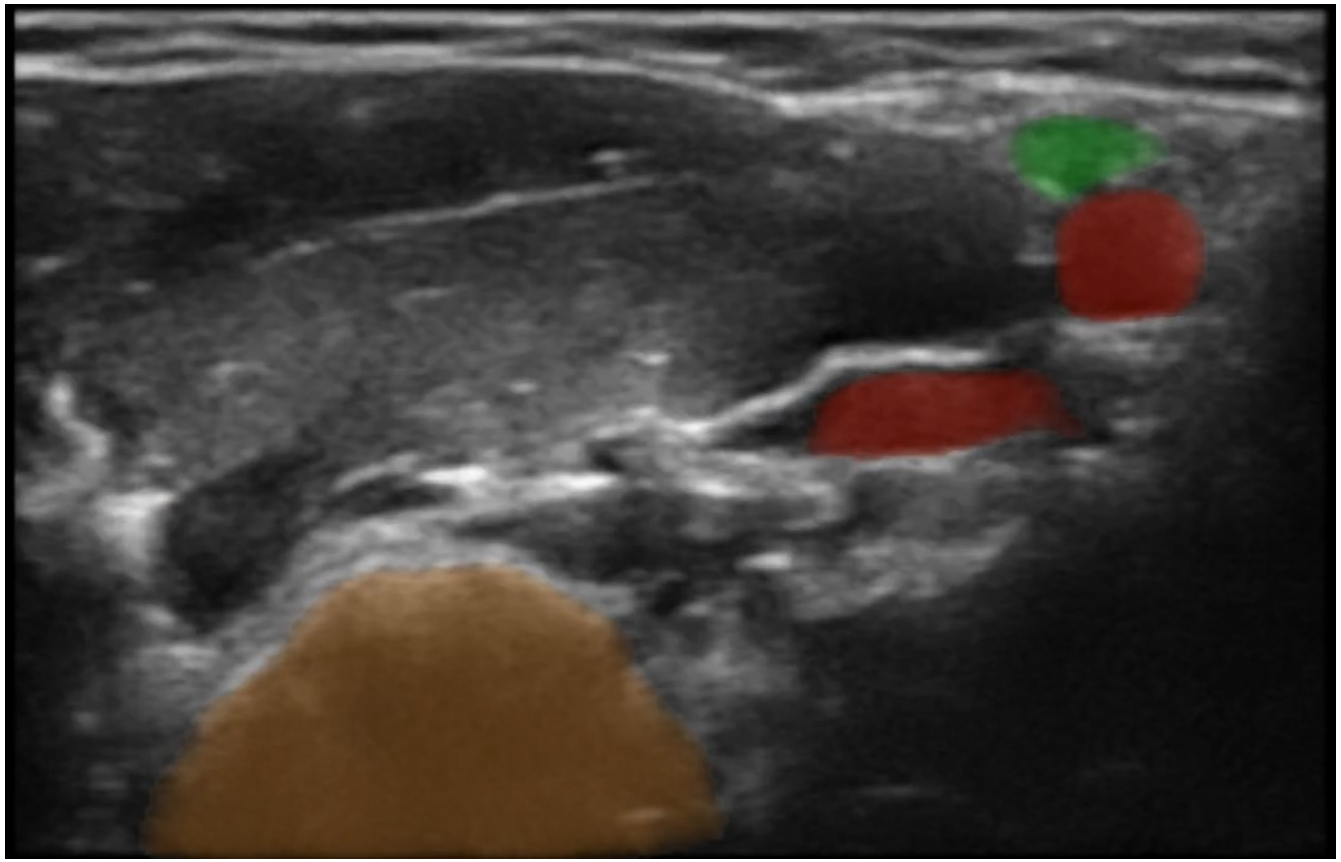
- Not all OpenCL implementations supports all image formats
  - In 1.2 the minimum to support was 4 channels (RGBA)
  - In 2.0 this was extended to both 1 channel images
  - Need to convert when moving data between host and CL device
- 3D image processing essential!
  - Need an extension to write to 3D images: cl_khr_3d_image_writes
    - Very happy to see it become core in CL 2, sad to see it become optional in CL 3

How FAST does visualization

- OpenGL 3.3
- Qt 5
- Multi-threading
  - Main thread – Visualization thread
  - Computation thread – Executes pipeline
  - Additional threads – Streaming ++

Image rendering

- Simple texture rendering



Volume rendering

- Ray casting in OpenCL
- Framebuffer object (FBO)

Geometry rendering

- Vertices, lines and triangles
- Vertex buffer object (VBO)
- Element buffer object (EBO)

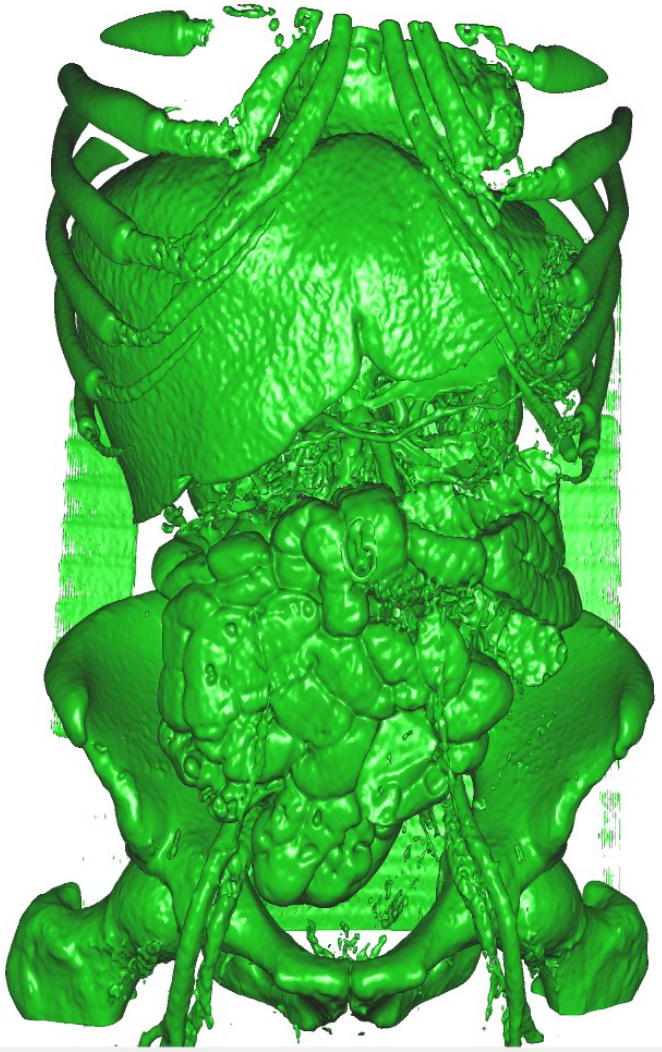
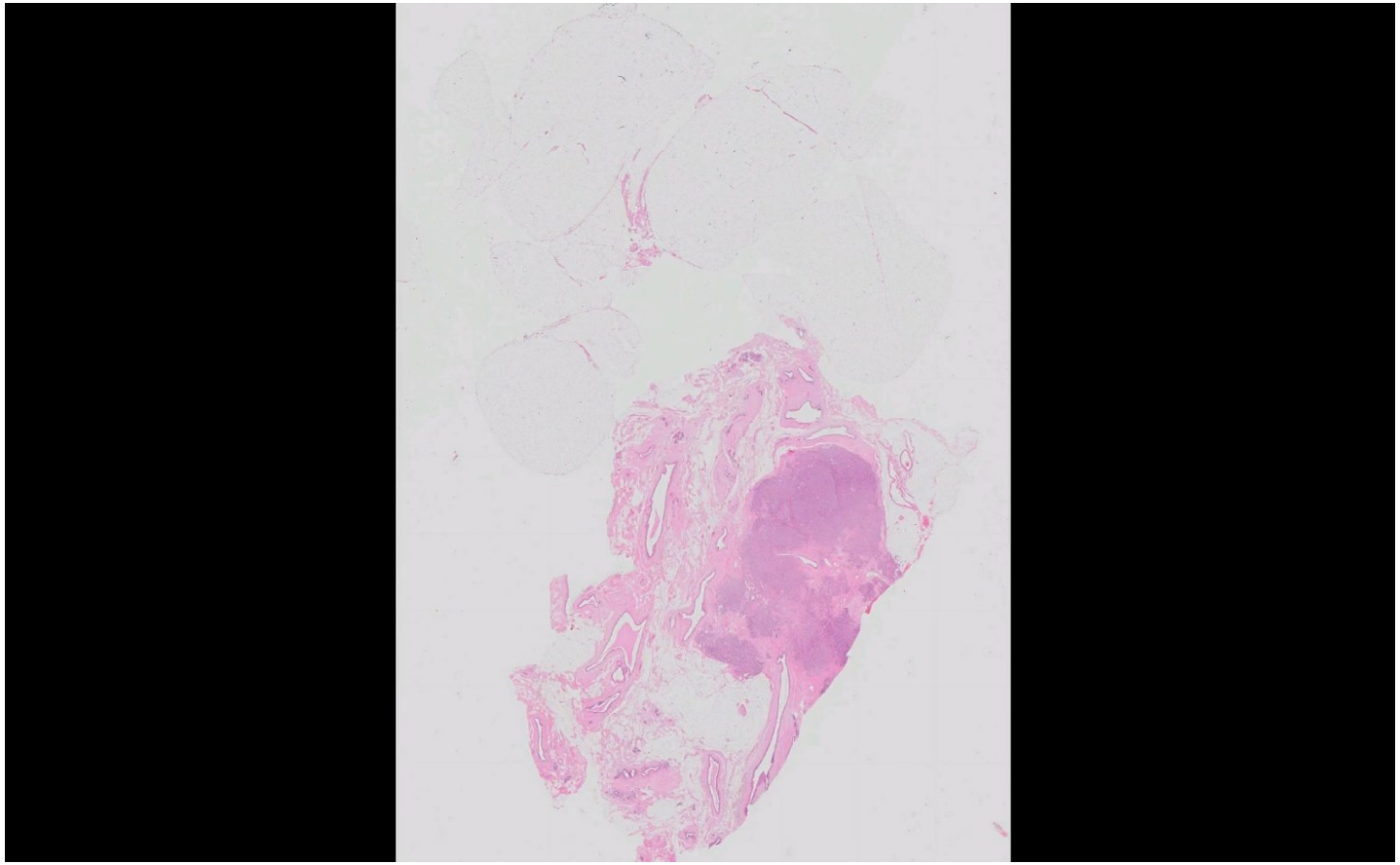


Image pyramid rendering

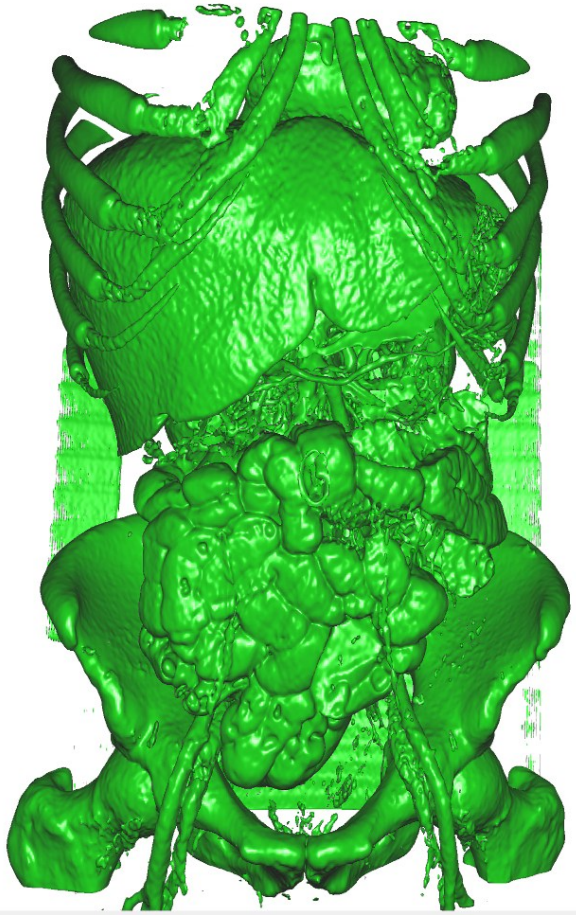
- Tiled image pyramids
- Virtual memory system
- Compressed GL textures



Video courtesy of André Pedersen and NTNU IKOM

OpenCL-OpenGL Interoperability

cl_khr_gl_sharing – Sharing textures/images and buffers



OpenCL-OpenGL interoperability

- Lack of support on Linux
  - Not working on Linux with NVIDIA. Hasn't been working the last 5 years.
    - *Xlib: extension "NV-GLX" missing on display*
  - Not implemented on Linux with Intel
    - https://github.com/intel/compute-runtime/issues/166
  - Need backup – CPU transfer

```
// Create OpenGL texture
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);
glTexImage2D(GL_TEXTURE_2D, 0, ..., width, height, ...);

// Create OpenCL Image from texture
auto imageGL = cl::ImageGL(context, ..., GL_TEXTURE_2D, 0, texture);

// Synchronization
std::vector<cl::Memory> v;
v.push_back(imageGL);
queue.enqueueAcquireGLObjects(&v);
// Do stuff
queue.enqueueReleaseGLObjects(&v);
```

OpenCL-OpenGL interoperability

- Why only GL -> CL? More useful with CL -> GL?
  - You normally do computations then visualization
  - OpenGL extension?
  - glTexImage2DFromCL(opencl-image)?

```
// Assume you have an OpenCL Image
auto imageCL = cl::Image(context, ....);

// Create OpenGL texture from CL image
glGenTextures(1, &texture);
glBindTexture(GL_TEXTURE_2D, texture);
glTexImage2DFromCL(imageCL);
// Visualize it!
```

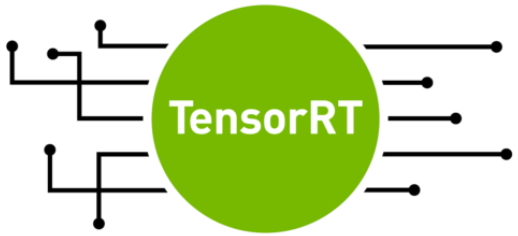
Deep learning

- Deep neural networks are the standard for most image processing tasks
  - FAST must support DNN inference
- Fragmented
  - Many different model formats
  - Many different frameworks
- FAST solution
  - Common interface
  - Runtime dynamic linking
    - OpenVINO
    - TensorRT
    - TensorFlow
  - Tensor data object
    - OpenCL Buffer
    - C++ pointer

```
auto importer = ImageFileImporter::New();
importer->setFilename("path/to/some_image.jpg");

auto network = SegmentationNetwork::New();
network->load("path/to/model.onnx");
network->setInputConnection(importer->getOutputPort());

auto renderer = SegmentationRenderer::New();
renderer->addInputConnection(network->getOutputPort());
```



Hopes for OpenCL in the future

- High performance **neural network inference** library running OpenCL on Intel, AMD, NVIDIA GPUs++
- Wider support for **cl_khr_3d_image_writes**
- Better **CL-GL interoperability** support - And the possibility of going from CL Image/Buffer to a GL Texture/Buffer
- OpenCL/OpenGL on Mac? Running **OpenCL on Metal**?

Summary

- Over several years created a high-performance framework for medical image computing and visualization
- OpenCL has been vital to its success
- Challenging to supports all major platforms (Windows, Linux, Mac) + (AMD, Intel, NVIDIA) while delivering high performance
  - Every extension used increases code complexity
- Try FAST yourself https://github.com/smistad/FAST

Thank you!

Contact: erik.smistad@ntnu.no

https://github.com/smistad/FAST