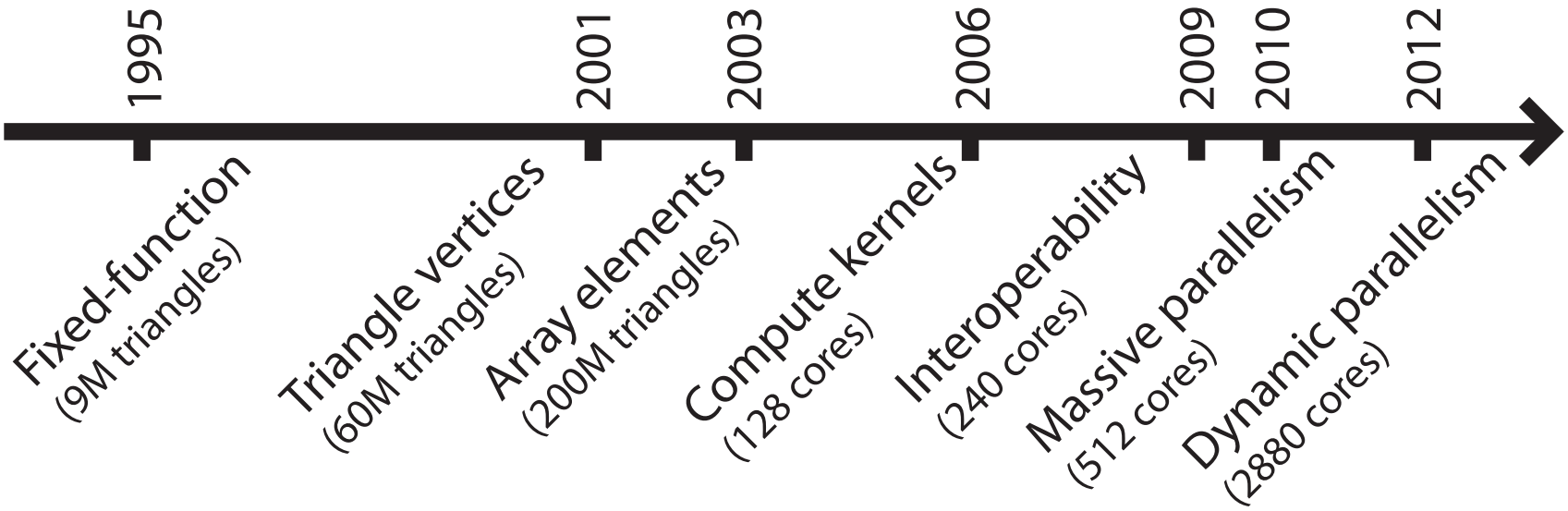
High GPU performance while the compiler handles the low level details

pros and cons of using a graphics pipeline

Jeppe Revall Frisvad

January 2013

Timeline on the programmability of the GPU



1995 Fixed-function rasterization pipeline in hardware.

2001 Vertex shaders (first programmable part of the pipeline).

2003 Fragment/pixel shaders (GPGPU).

2006 Unified shaders (CUDA) and geometry shaders.

2008 High level GPU programming in MATLAB with Jacket.

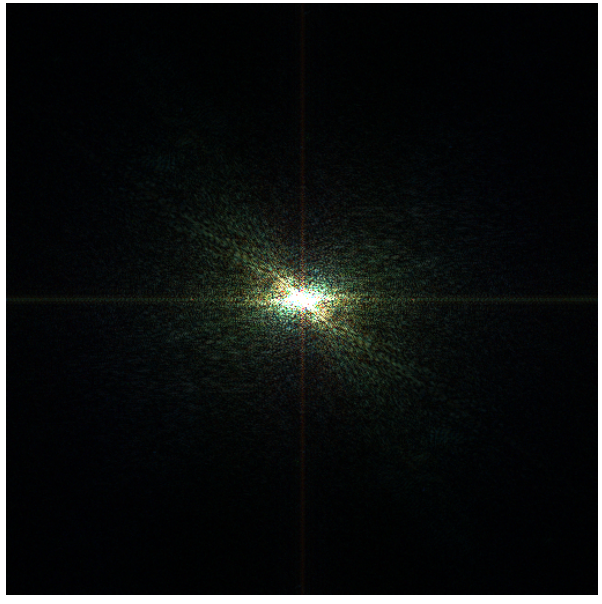
2009 Compute shaders (interoperability) and tesselation shaders.

2010 Programmable ray tracing pipeline on the GPU (OptiX).

2012 Dynamic parallelism (threads spawn threads).

Trade-offs between performance and ease of use

- MATLAB is a simple and easy way to use the GPU.
  How much efficiency do we lose?
- CUDA offers both high and low level functionality.
  It is easy to get started, but hard to get good performance.
- GPGPU requires basic knowledge of graphics programming.
  Harder to get started, easier to get good performance.
- Let us explore the validity of these statements.
- Case study: The full colour FFT of Lenna.

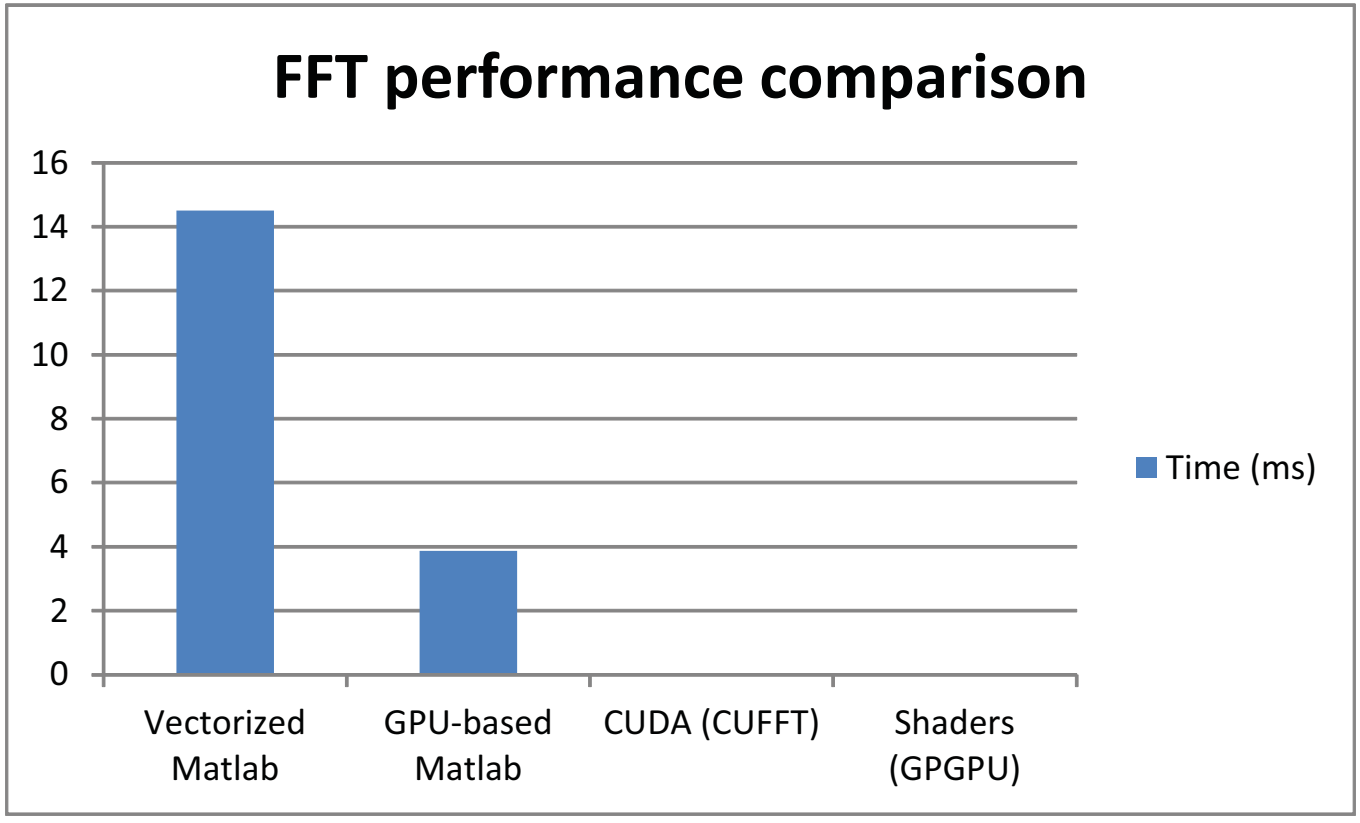
The full color FFT of Lenna in MATLAB

```
img = imread('Lenna.png');
img_size = size(img);
s = sqrt(img_size(1)*img_size(2));
float_img = (single(img) + 0.5)/256;

tic;
fft_img = fft2(float_img);
norm_img = sqrt(fft_img.*conj(fft_img));
norm_img = rgb_fftshift(clamp(norm_img/s, 0, 1));
toc;

image(norm_img);
```

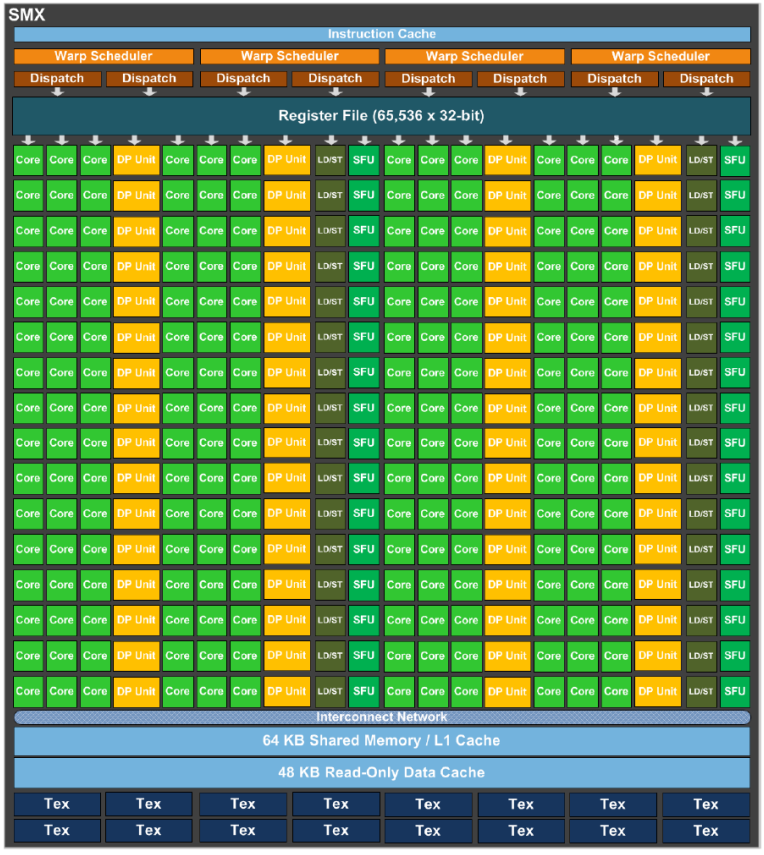
GPU computing in MATLAB

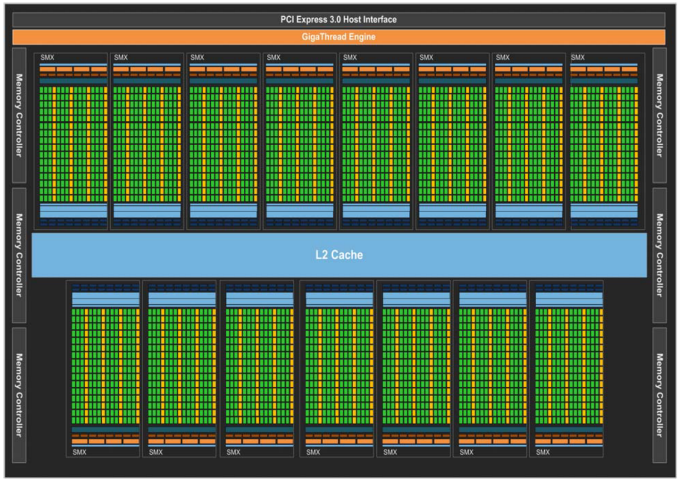
```
img = imread('Lenna.png');
img_size = size(img);
s = sqrt(img_size(1)*img_size(2));
float_img = (single(img) + 0.5)/256;
gpu_img = gpuArray(float_img);
tic;
fft_img = fft2(gpu_img);
norm_img = sqrt(fft_img.*conj(fft_img));
norm_img = rgb_fftshift(clamp(norm_img/s, 0, 1));
toc;
result = gather(gpu_img);
image(result);
```

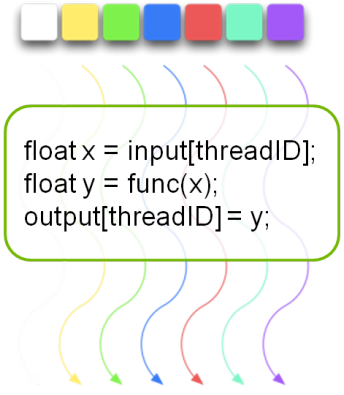
From vectorized MATLAB to GPU-based MATLAB

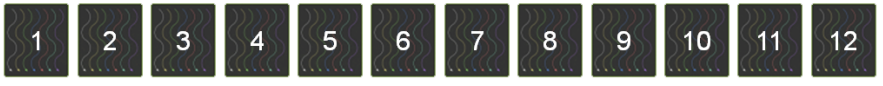


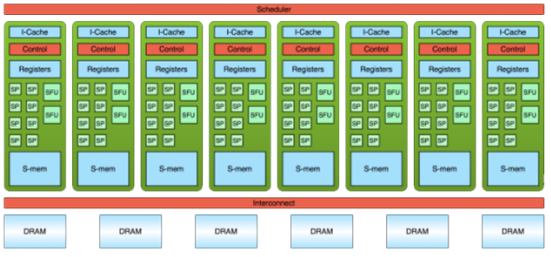
GPU architecture (©NVIDIA)

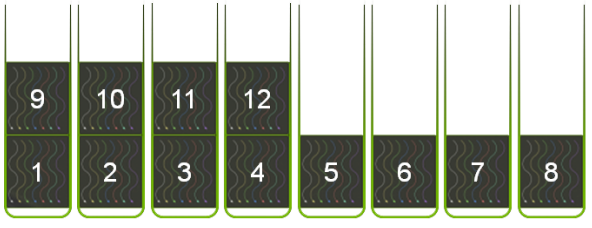












CUDA C version using CUFFT - inititalization

```
// Load image from file
int width, height, channels;
unsigned char* data = SOIL_load_image("Lenna.png",
     &width, &height, &channels, SOIL_LOAD_RGB);

// Declare variables for CUDA code
unsigned int size = width*height;
unsigned int half_width = width/2 + 1;
unsigned char* result
     = new unsigned char[size*channels];
uchar3* img = 0;
cufftReal* float_img = 0;
cufftComplex* fft_img = 0;
uchar3* norm_img = 0;
```

CUDA C version using CUFFT - allocation and transfer

```
// Allocate GPU buffers

cudaMalloc((void**)&img, size*sizeof(uchar3));

cudaMalloc((void**)&float_img,
     size*sizeof(cufftReal));

cudaMalloc((void**)&fft_img,
     half_width*height*sizeof(cufftComplex));

cudaMalloc((void**)&norm_img, size*sizeof(uchar3));


// Copy input image from host to GPU

cudaMemcpy(img, data, size*sizeof(uchar3),
     cudaMemcpyHostToDevice);


// Create FFT plan

cufftHandle planR2C;

cufftPlan2d(&planR2C, width, height, CUFFT_R2C);
```

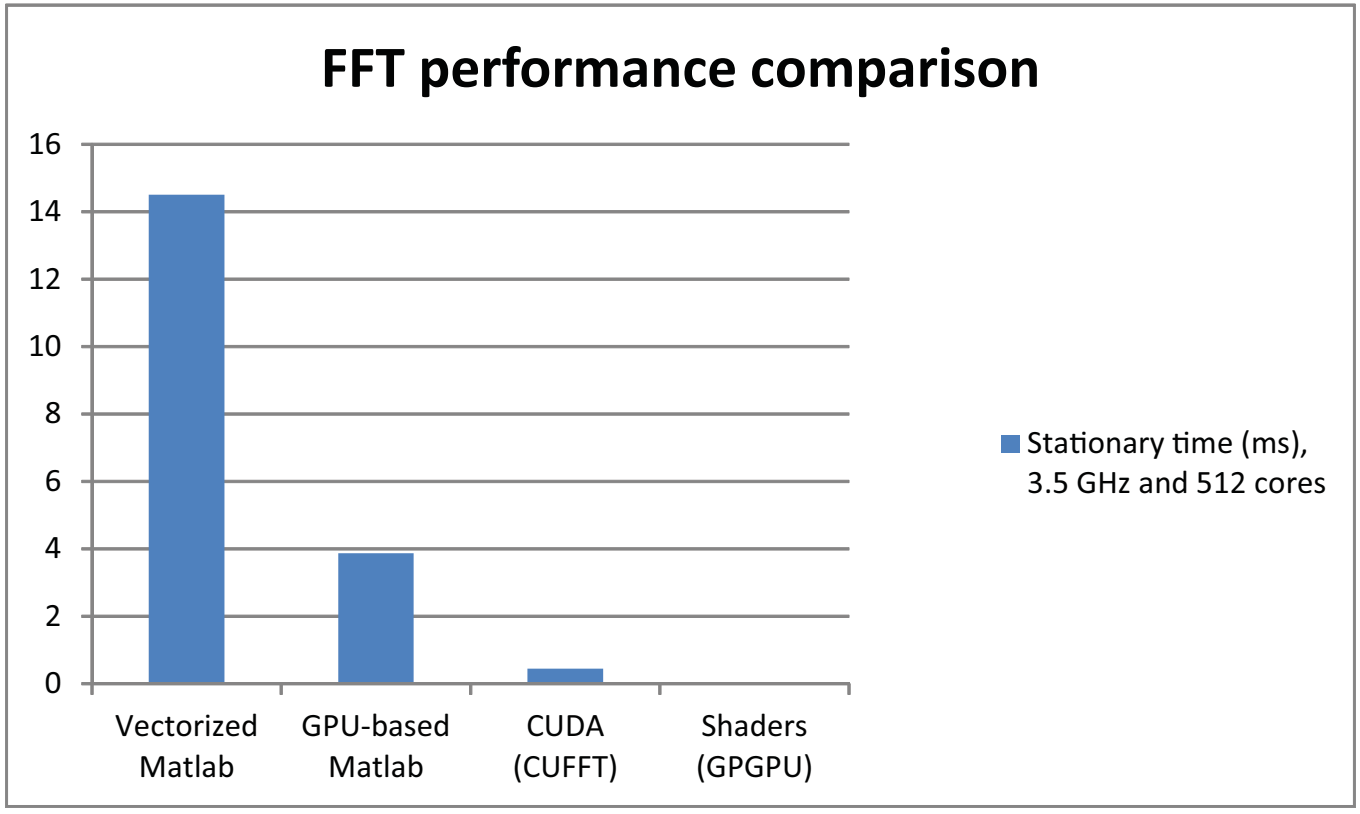
CUDA C version using CUFFT - kernel invocation

```
timer.start();
// Launch kernels and perform FFT on the GPU
// with one block for each row of pixels.
uchar3_to_floatR<<<height, width>>>(float_img, img);
cufftExecR2C(planR2C, float_img, fft_img);
complexR_to_uchar3<<<height, half_width>>>(norm_img,
     fft_img, width, height);
⋮
(same for green (G) and blue (B))
⋮
cudaDeviceSynchronize();
timer.stop();
```

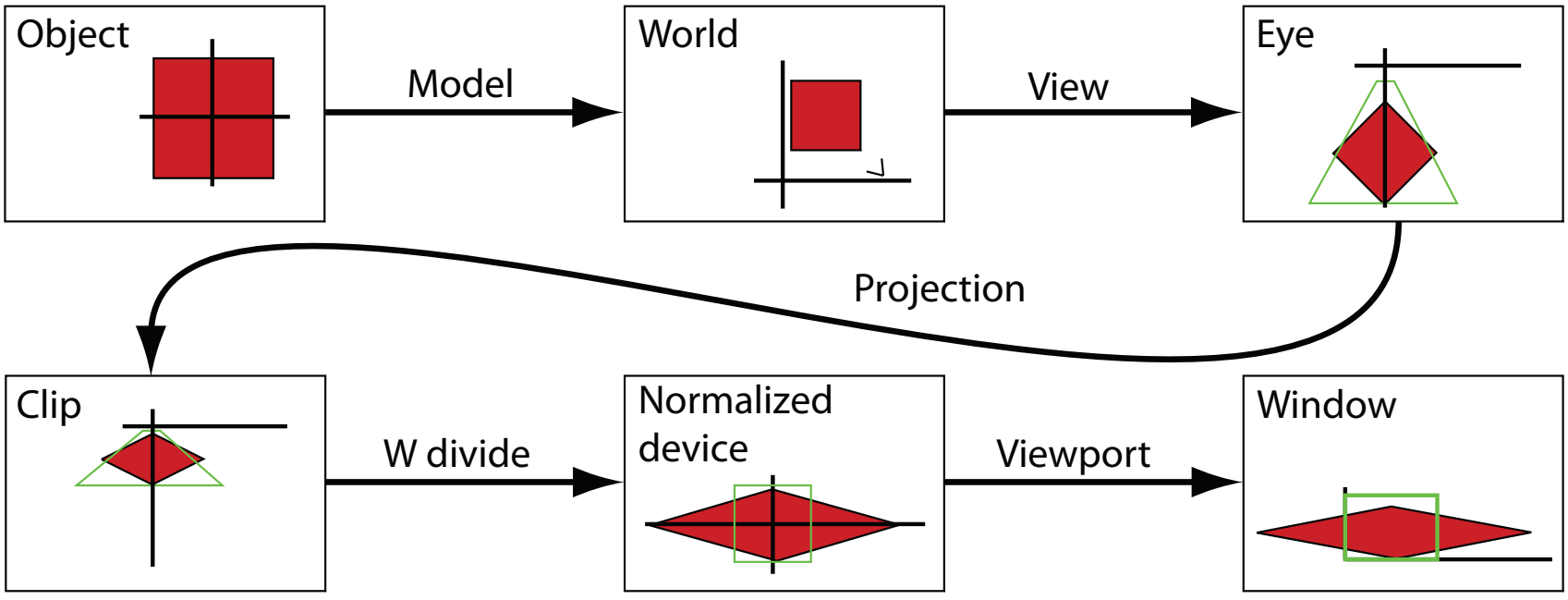
CUDA C version using CUFFT - finalization

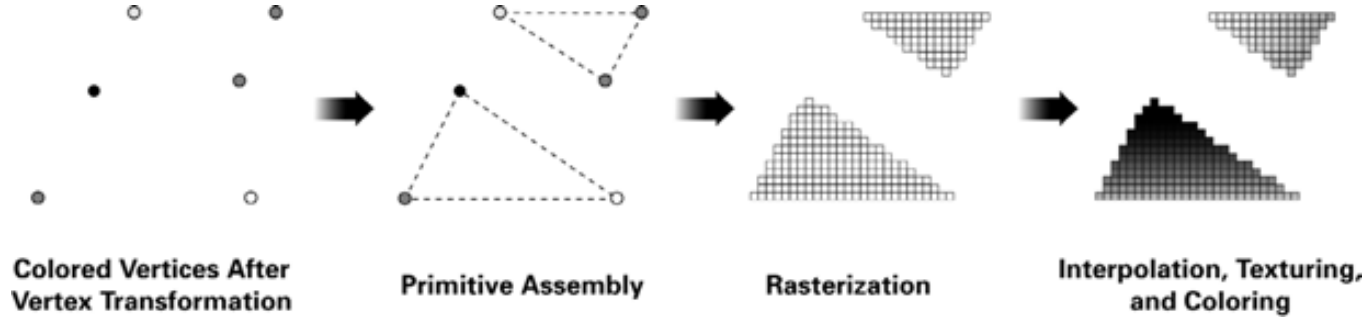
```
// Copy output image from GPU to host
cudaMemcpy(result, norm_img, size*sizeof(uchar3),
     cudaMemcpyDeviceToHost);
stbi_write_png("fft.png", width, height, channels,
     result, width*channels);


// Clean up (free memory)
cufftDestroy(planR2C);
cudaFree(img);
cudaFree(float_img);
cudaFree(fft_img);
cudaFree(norm_img);
delete [] result;
```

From vectorized MATLAB to GPU-based MATLAB

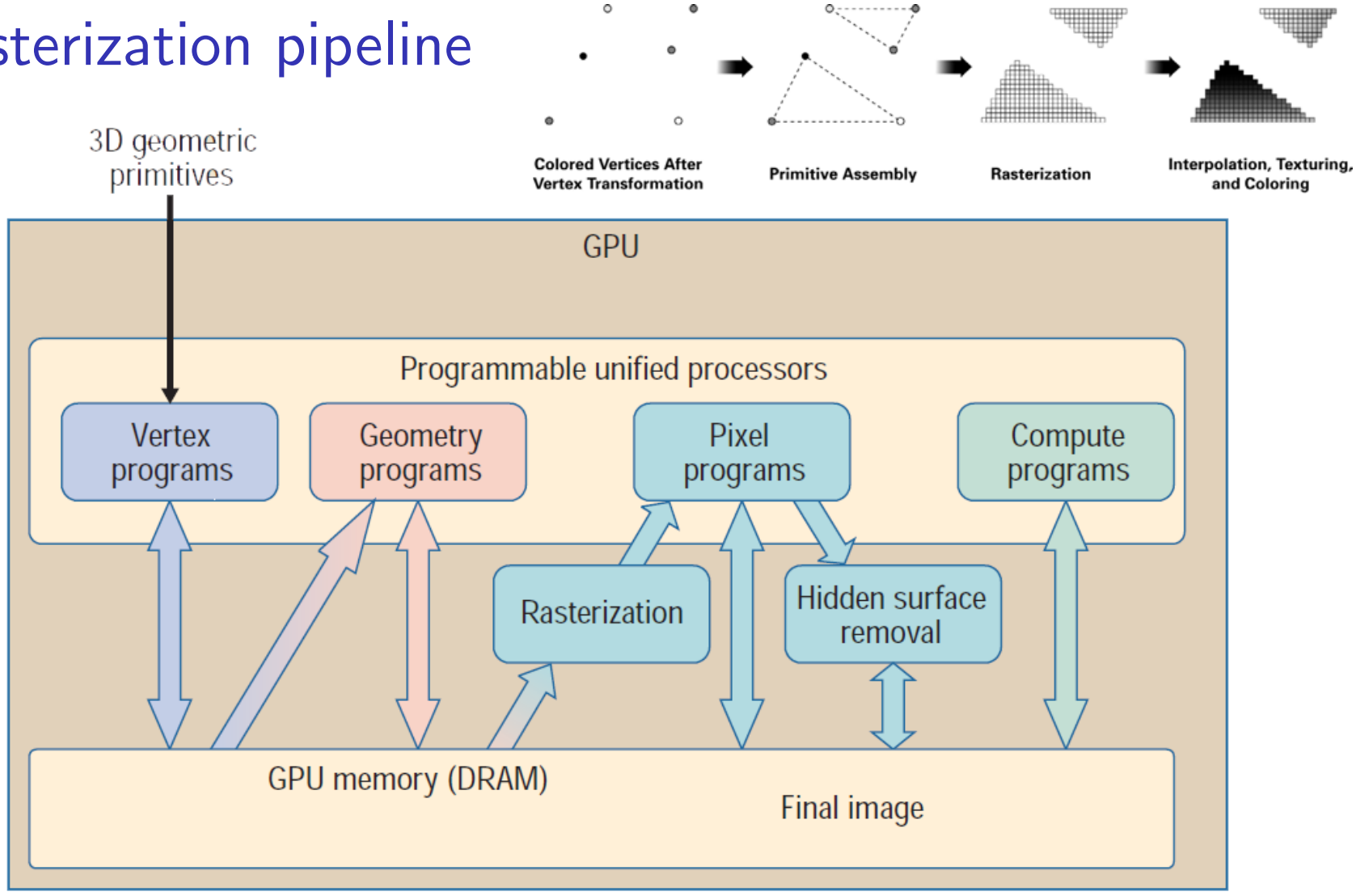


Rasterization pipeline - classical version





Rasterization pipeline

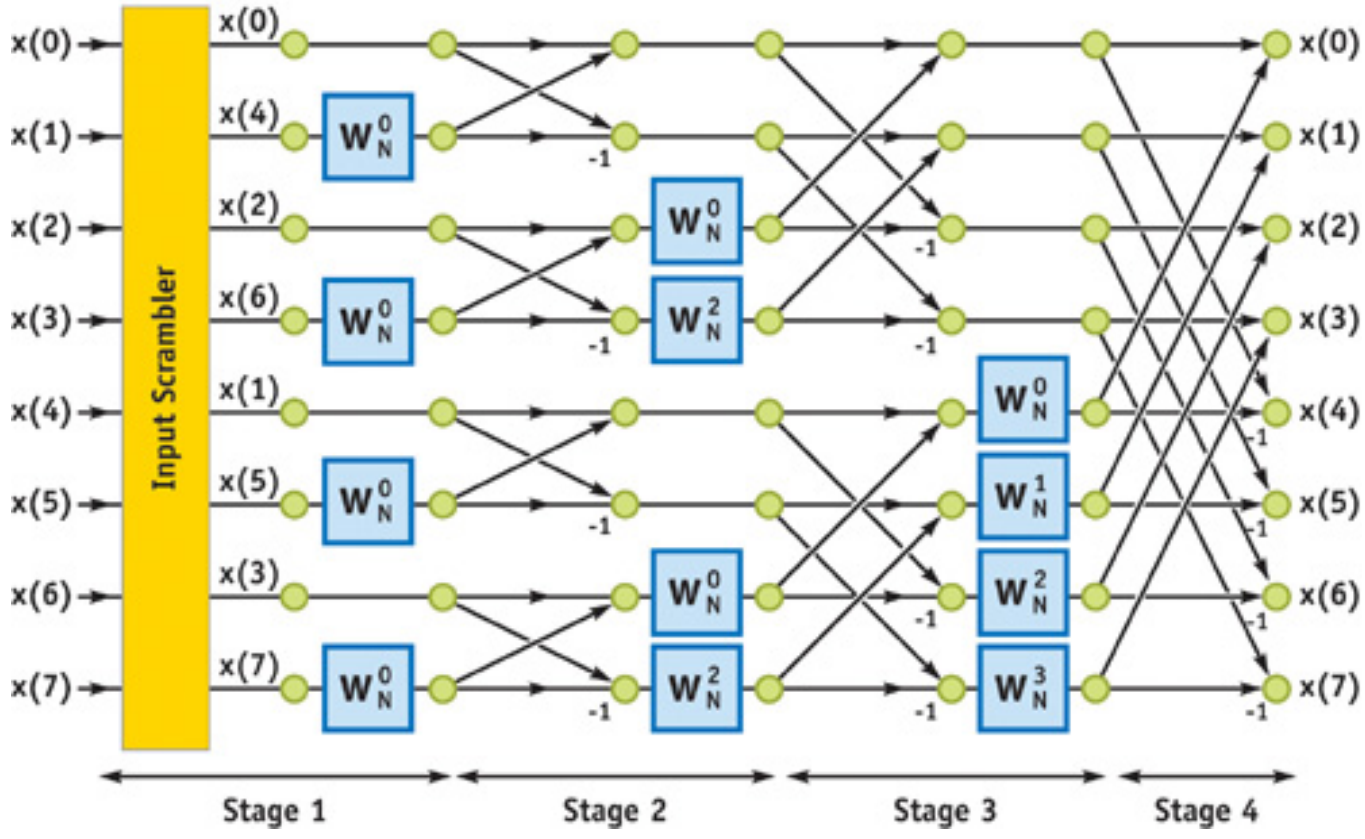


- The rasterization pipeline is still available in GPUs.

References

- Fernando, R., and Kilgard, M. J. Introduction. In *The Cg Tutorial: The Definitive guide to Programmable Real-Time Graphics*, Chapter 1, Addison-Wesley, 2003.
- Luebke, D., and Humphreys, G. How GPUs work. *Computer 40*(2), pp. 96–100, February 2007.

FFT on the GPU using shaders



- Only $2\log_2(N)$ passes for two 2D FFTs (N is width).
- Scrambler indices and weights in small 1D textures.

Shader version (GPGPU) - creating a context

```
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitWindowSize(width, height);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA |
        GLUT_DEPTH | GLUT_ALPHA);
    glutCreateWindow("GPU FFT demo");
    glewInit();
    // Load image from file
    int width, height, channels;
    unsigned char* data = SOIL_load_image("Lenna.png",
        &width, &height, &channels, SOIL_LOAD_RGB);
    // Transfer image to texture memory
    SOIL_create_OGL_texture(data, width, height,
        channels, SOIL_CREATE_NEW_ID,
        SOIL_FLAG_INVERT_Y);
    :
    return 0;
}
```

Shader version (GPGPU) - drawing triangles

```
GLuint source_list = glGenLists(1);
glNewList(source_list, GL_COMPILE);
glEnable(GL_TEXTURE_2D);
glBindTexture(GL_TEXTURE_2D, source_tex);
glBegin(GL_POLYGON);
    glTexCoord2f(0.0f, 0.0f);
    glVertex2f(-1.0f, -1.0f);
    glTexCoord2f(1.0f, 0.0f);
    glVertex2f(1.0f, -1.0f);
    glTexCoord2f(1.0f, 1.0f);
    glVertex2f(1.0f, 1.0f);
    glTexCoord2f(0.0f, 1.0f);
    glVertex2f(-1.0f, 1.0f);
glEnd();
glDisable(GL_TEXTURE_2D);
glEndList();
```

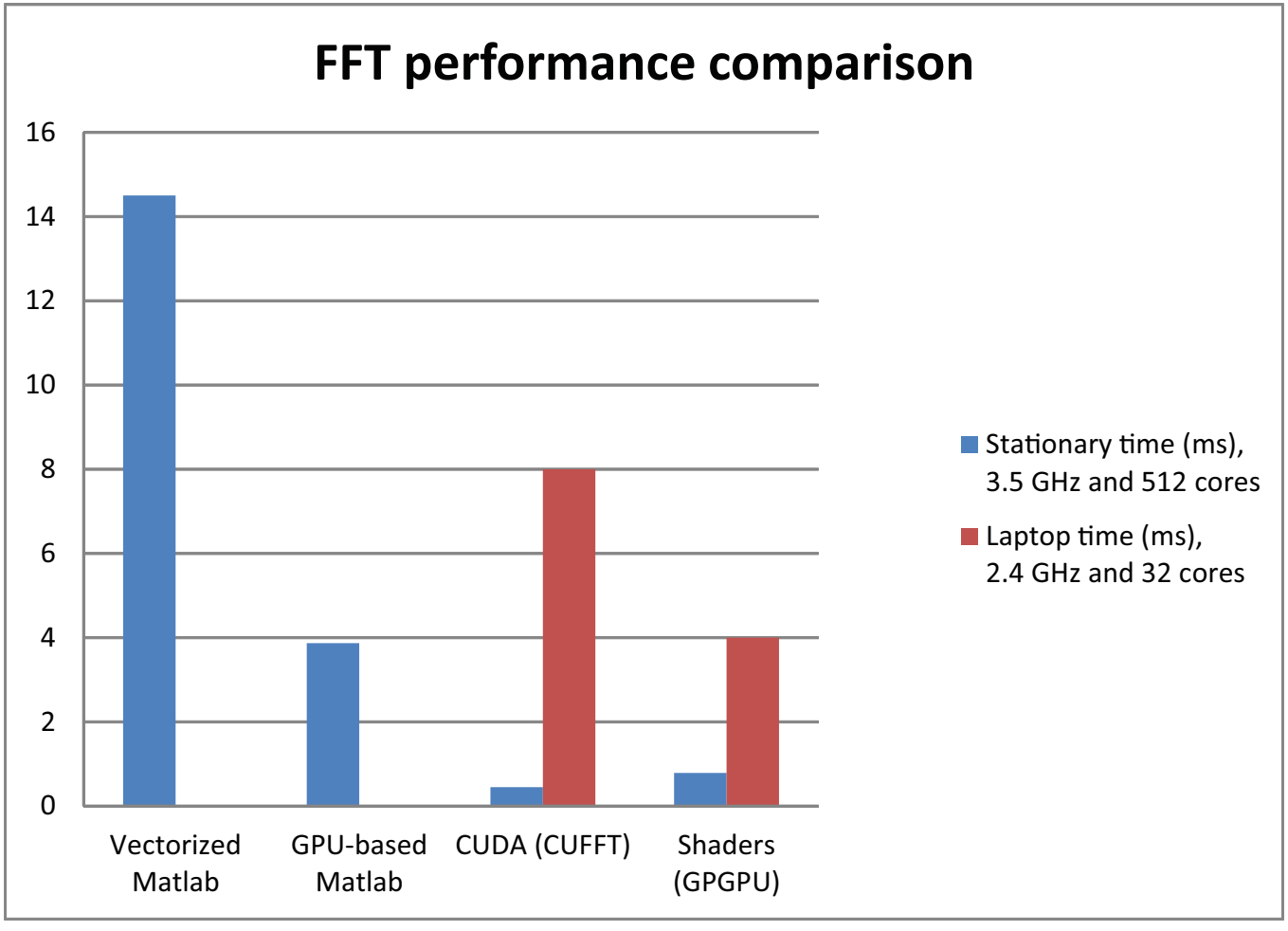
Shader version (GPGPU) - loading shaders

```
// Create shaders
GLuint vs = glCreateShader(GL_VERTEX_SHADER);
GLuint fs = glCreateShader(GL_FRAGMENT_SHADER);
// Load source code strings into shaders
glShaderSource(vs, 1, &vert_shader, 0);
glShaderSource(fs, 1, &frag_shader, 0);
// Compile shaders
glCompileShader(vs); glCompileShader(fs);
// Create a program and attach the shaders
GLuint prog = glCreateProgram();
glAttachShader(prog, vs);
glAttachShader(prog, fs);
glLinkProgram(prog); // Link the program
```

Shader version (GPGPU) - using the FFT

```
const float scale = 1.0f/sqrt(width*height);
FFT* fft = new FFT(width, height);
timer.start();
fft->set_input(draw_fft_source_rb);
fft->do_fft();
glBlendFunc(GL_ONE, GL_ONE);
glEnable(GL_BLEND);
fft->draw_output(scale, 0.0f, 0.0f, 1);
fft->draw_output(0.0f, 0.0f, scale, 2);
glDisable(GL_BLEND);
fft->set_input(draw_fft_source_g);
fft->redraw_input();
fft->do_fft();
glEnable(GL_BLEND);
fft->draw_output(0.0f, scale, 0.0f);
glDisable(GL_BLEND);
timer.stop();
```

Performance

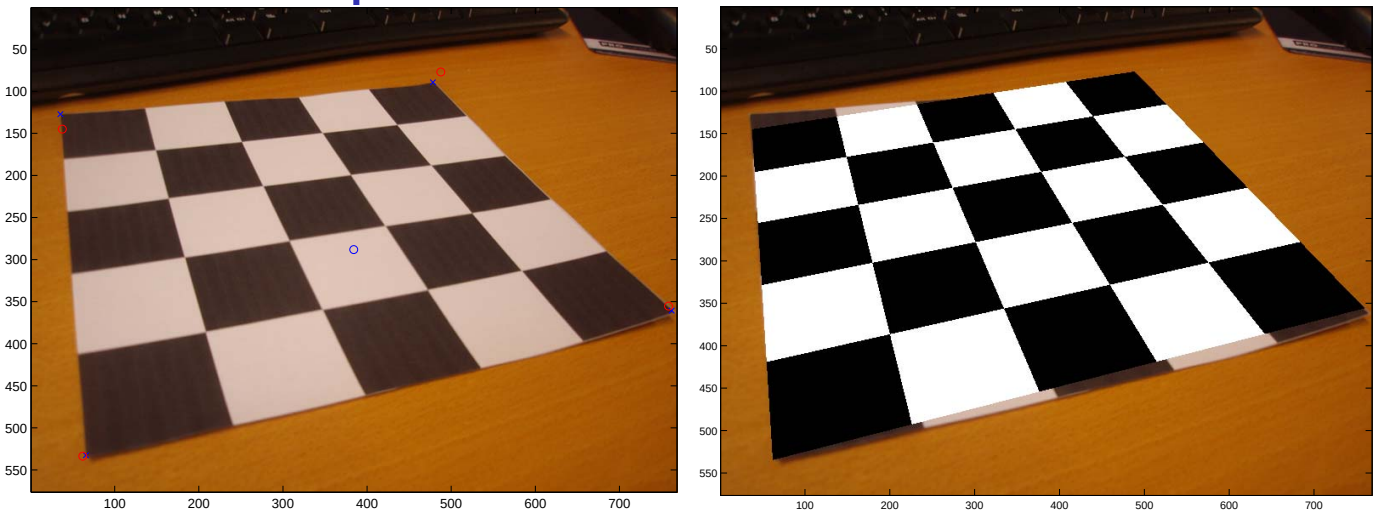


- Are newer GPUs better at CUDA or were the CUDA magic numbers better suited for the 512 cores?

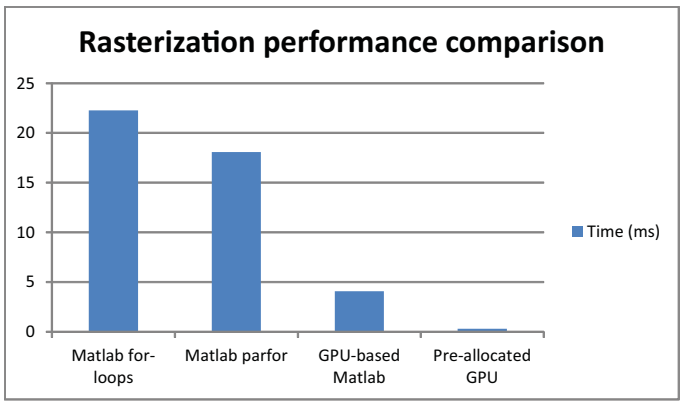
What were the difficulties?

- MATLAB
  - None.
- CUDA (in summary)
  - Two pass compilation (.cu files and the NVCC compiler).
  - CUDA libraries dependencies.
  - Magic numbers for thread/block allocation.
  - Easy to make inefficient code (non-coalesced memory access).
  - Must write kernels (syntax needs learning).
- Shaders (in summary)
  - Graphics context needed (library dependency).
  - OpenGL extensions need detection (library dependency).
  - Algorithms run through the graphics pipeline.
  - Runtime compilation.
  - Must write shaders (syntax needs learning).

Rasterization example

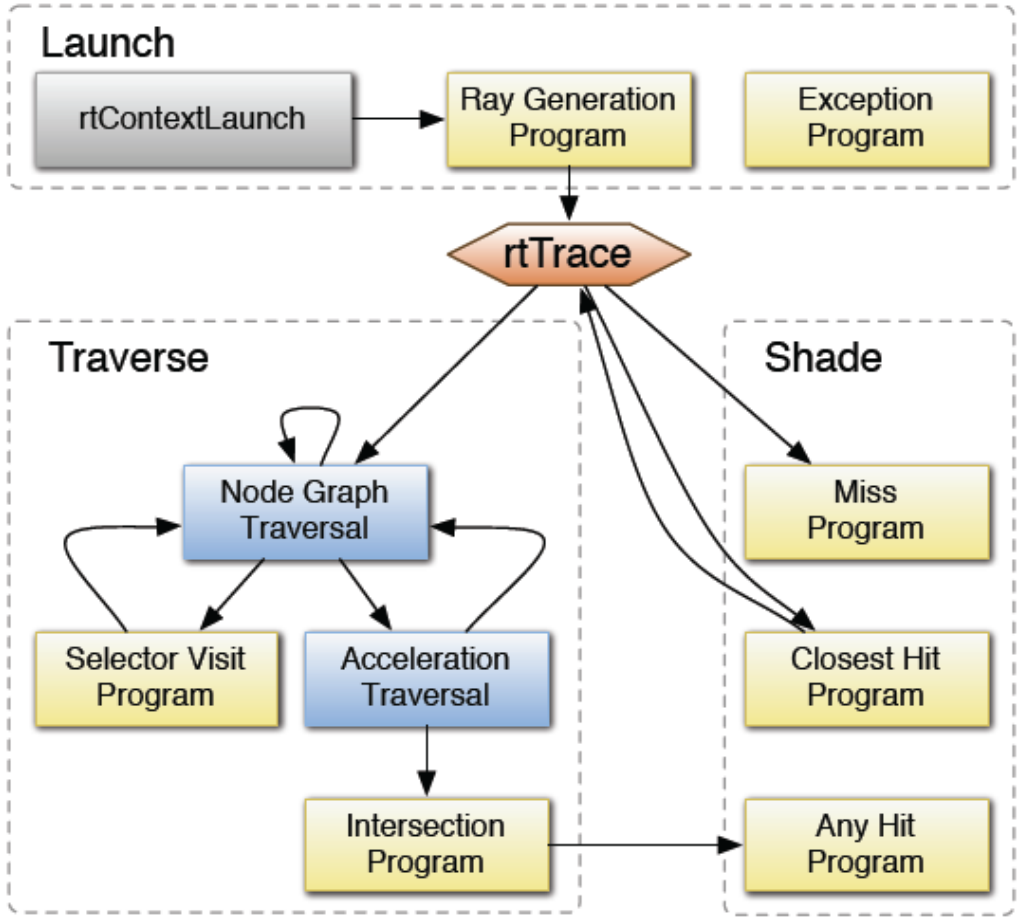


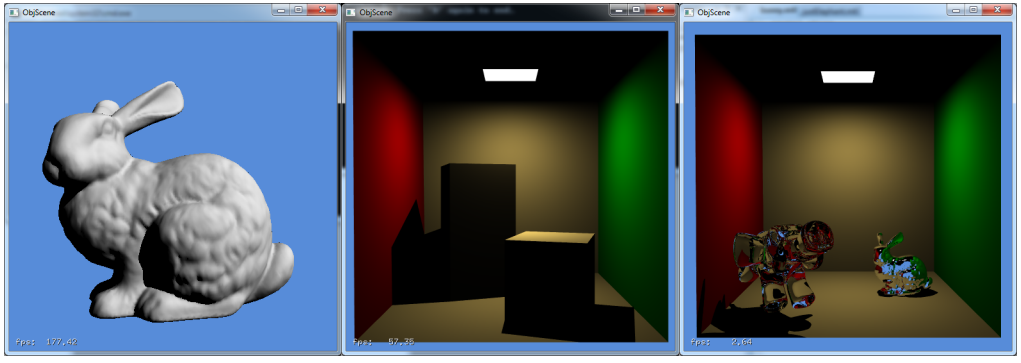
```
≫ render
Elapsed time is 22.266896 seconds.
≫ render
Starting matlabpool using the 'local' profile ...  connected to 6 workers.
Sending a stop signal to all the workers ...  stopped.
Elapsed time is 18.082566 seconds.
≫ gpu_render
Elapsed time is 4.079594 seconds.
≫ gpu_render
Elapsed time is 0.306040 seconds.
```



- Frame rate using graphics pipeline: >5000 frames per second.

GPU ray tracing pipeline (OptiX)





Interoperability

- Compute shaders
  - Rasterization-computing interop.
  - Compiler handles thread/block allocation.
- OptiX
  - Interop functionality for OpenGL/DirectX and CUDA.
  - Ray tracing-rasterization-computing interop.
  - Thread/block allocation needed for computing kernels.

- Downside: Interop requires a graphics context.

- **Thank you for your attention.**