INFO-H503

Real-time 3D graphics processing and GPGPU imaging

Projects

Overview projects

	Equ/recipes available	C-Code available	CUDA code available
Stereo disparity	Y	Y	N
Light Field depth estimation	Y	N	N
Face detector	Y (in CUDA)	Y	N
Bilateral filtering	Y	Y	Y
Mutual Information Registration	Y	Y	N
2D wavelet transform	Y	Matlab	OpenCL
Cloth simulation	Y	Y/N	N
Sound wave propagation	Y	N	Y/N
3D wavelet transform	Y	N	N
Least square 3D object fitting	Y (in CUDA)	N	N
Surface normal estimation	Y	N	N
Almost rigid deformations	Y	Y	N
Geodesic distances	Y	N	N
Simple ray tracer	Y	Y	Y/N
Volume ray tracer	Y	N	Y

Attention points

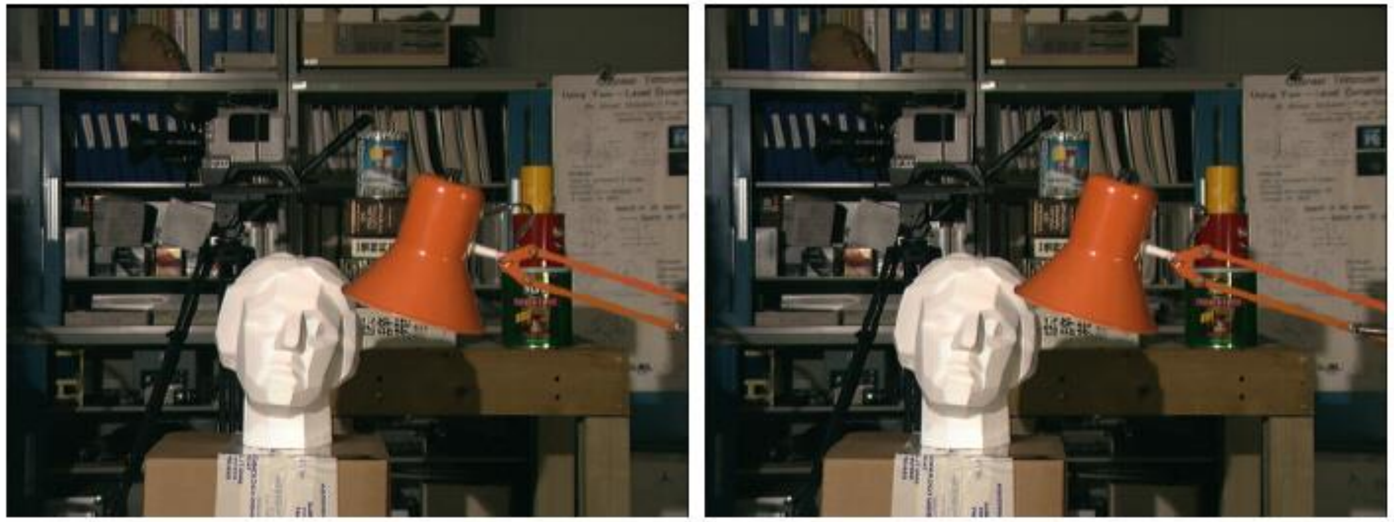
- Understand scientific papers
- Papers are available, sometimes also C/C++ and even CUDA code (in simplified version, as guideline)
- Image processing: in/out = file, real-time rendering is optional
- 3D graphics: minimal OpenGL 3D rendering needed
- Difficult projects: 1-2 students with some modules chosen
- CUDA mandatory, OpenCL optional
- CPU and GPU timing needed
- GPU optimizations needed
- Gradual presentations

Depth estimation

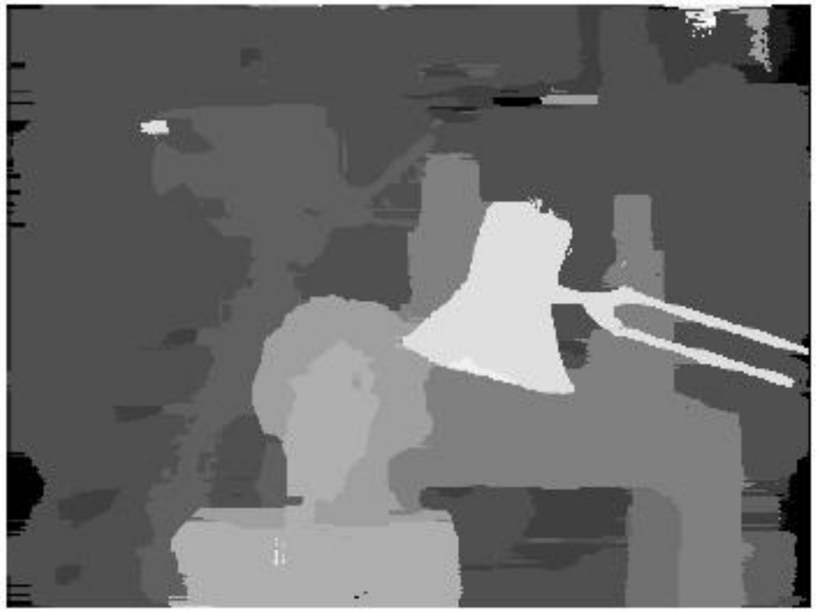
Stereo disparity

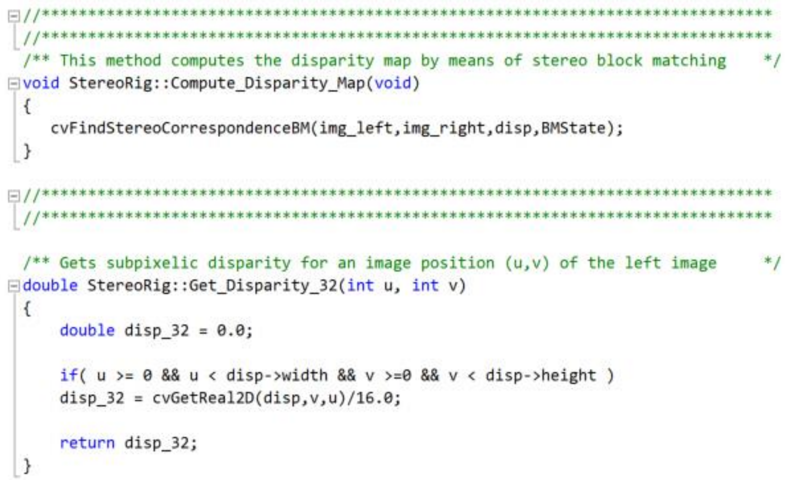
Stereo Disparity through Cost Aggregation with Guided Filter

Pauline Tan[1], Pascal Monasse[2]



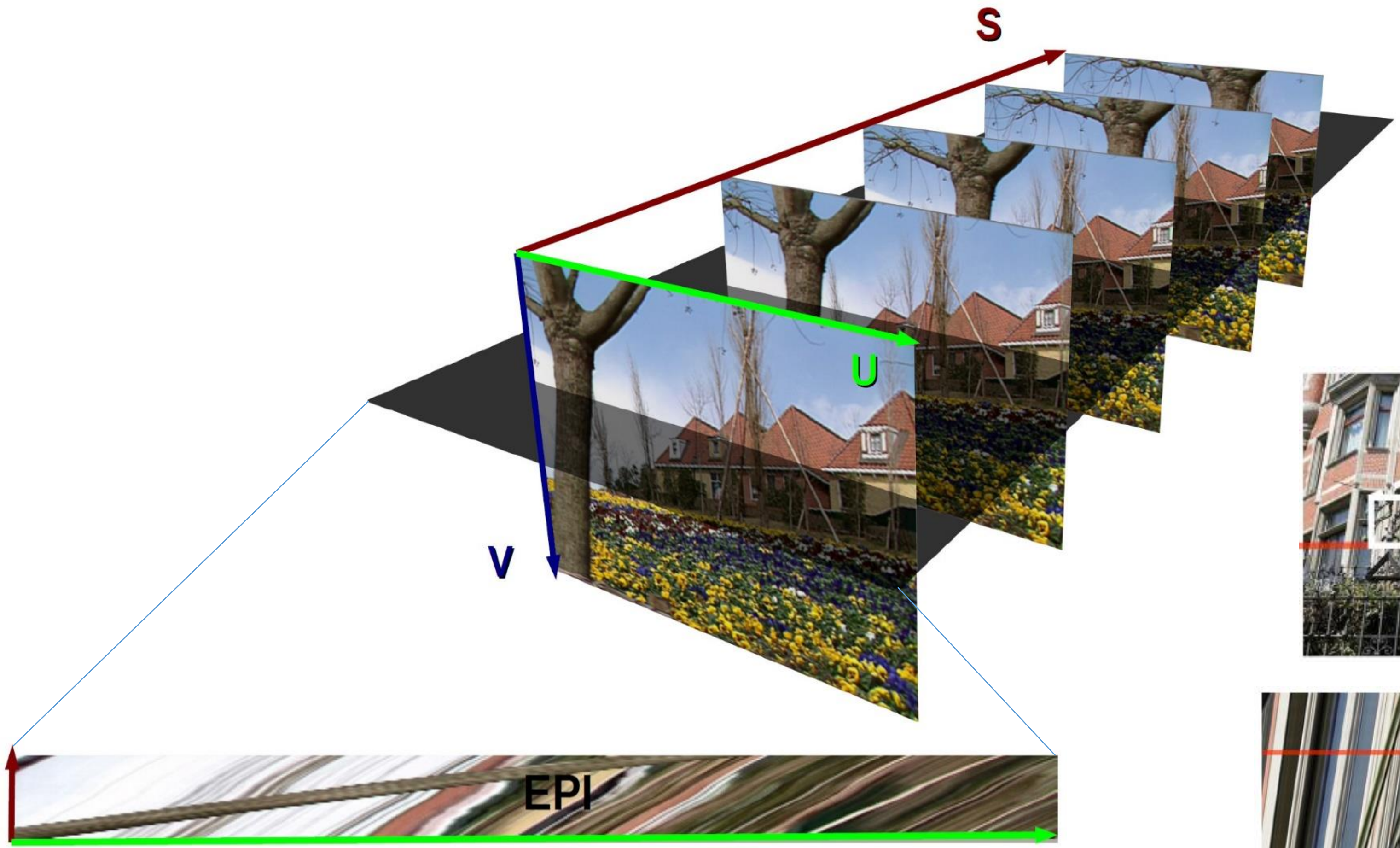
(a) Left (reference) image (b) Right (target) image



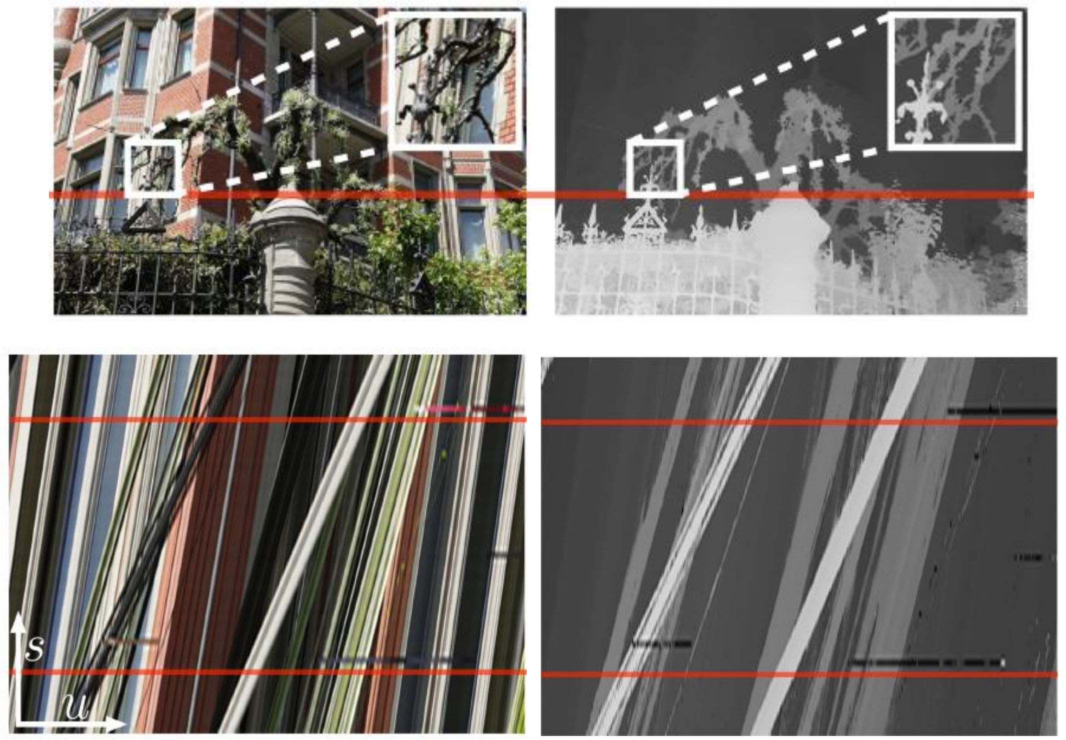


```
//*************************************************************************
//*************************************************************************
/** This method computes the disparity map by means of stereo block matching    */
void StereoRig::Compute_Disparity_Map(void)
{
   cvFindStereoCorrespondenceBM(img_left,img_right,disp,BMState);
}

//*************************************************************************
//*************************************************************************

/** Gets subpixelic disparity for an image position (u,v) of the left image     */
double StereoRig::Get_Disparity_32(int u, int v)
{
    double disp_32 = 0.0;

    if( u >= 0 && u < disp->width && v >=0 && v < disp->height )
    disp_32 = cvGetReal2D(disp,v,u)/16.0;

    return disp_32;
}
```

Light Field Depth Estimation (1-2 students)



Courtesy of UHasselt

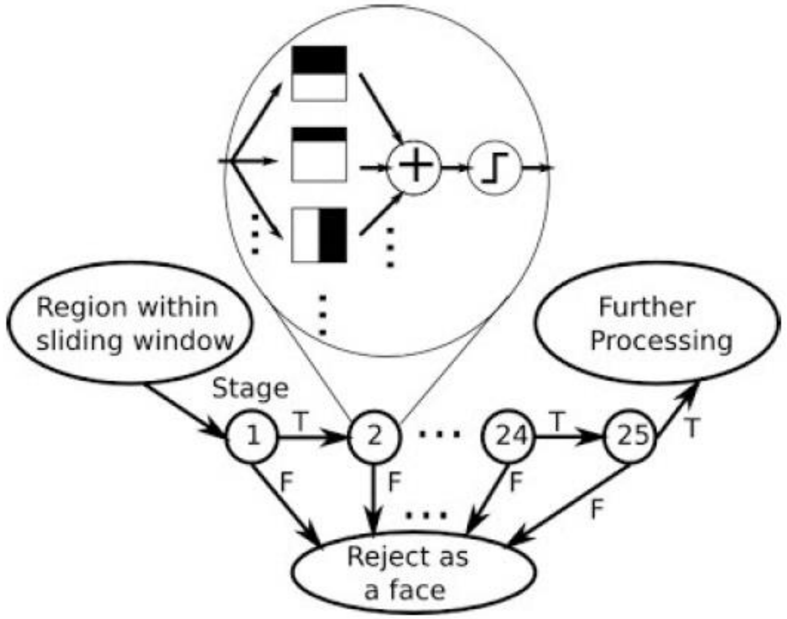


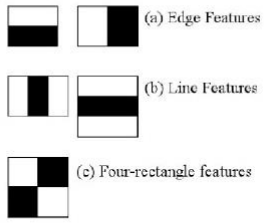
Feature detection and Filtering

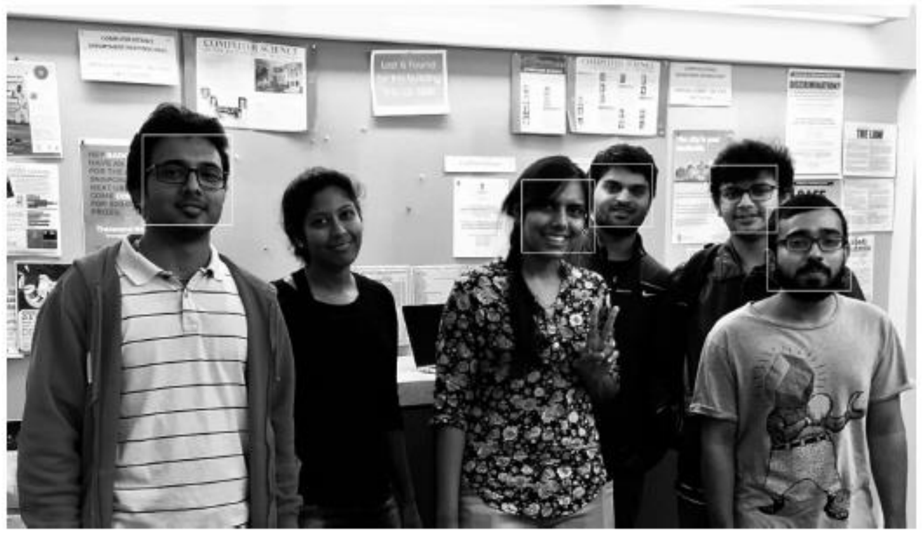
Face detector (1-2 students)

An Efficient GPGPU Implementation of Viola-Jones Classifier Based Face Detection Algorithm

Vinay Gangadhar Sharmila Shridhar Ram Sai Manoj Bamdhamravuri
{gangadhar, sshridhar, bamdhamravur}@wisc.edu







Bilateral filtering

$$GC[I]_{\mathbf{p}} = \sum_{\mathbf{q} \in \mathcal{S}} G_\sigma(\|\mathbf{p} - \mathbf{q}\|)\, I_{\mathbf{q}},$$

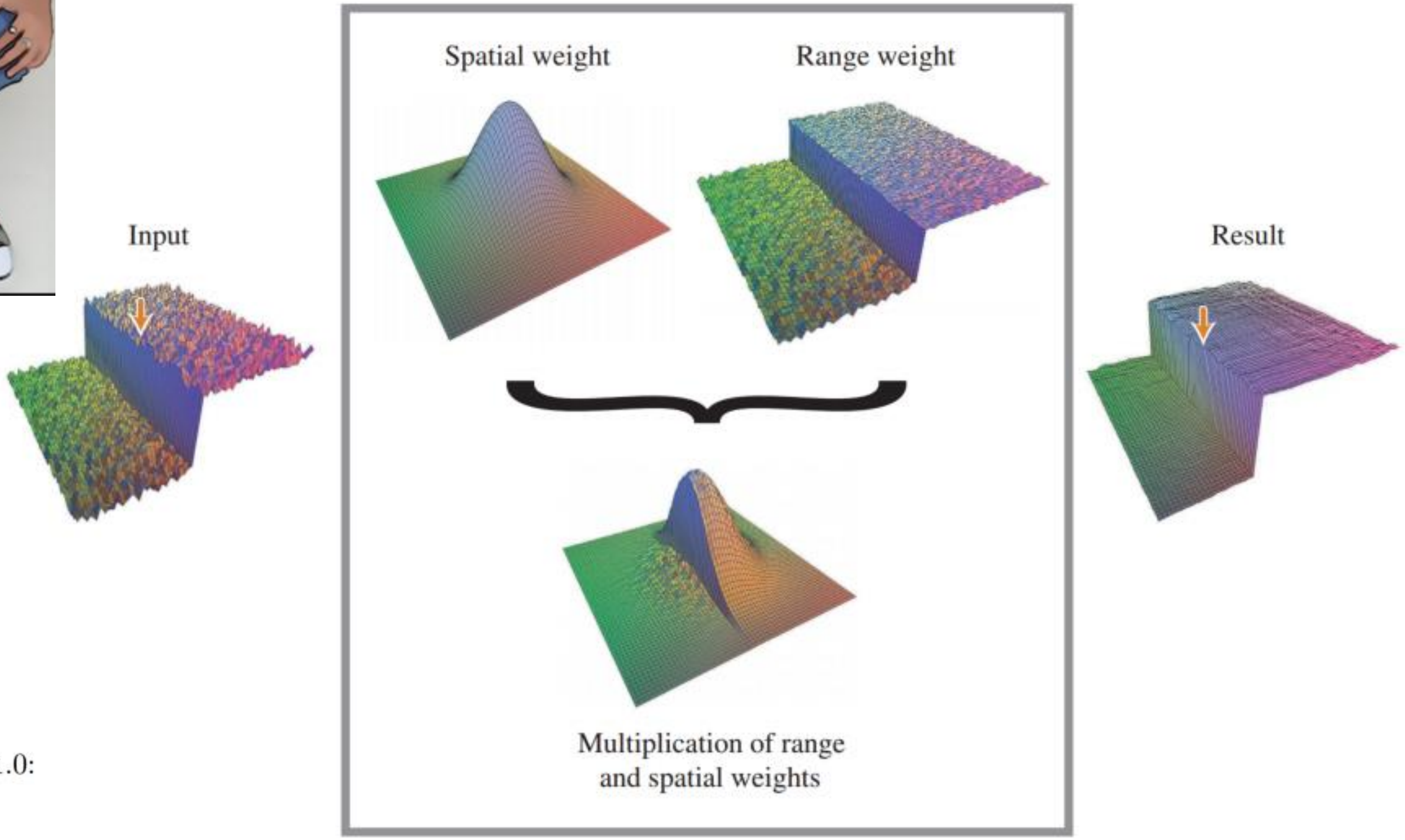
where $G_\sigma(x)$ denotes the 2D Gaussian kernel (see Figure 2.1):

$$G_\sigma(x) = \frac{1}{2\pi\,\sigma^2} \exp\left(-\frac{x^2}{2\sigma^2}\right).$$

$$BF[I]_{\mathbf{p}} = \frac{1}{W_{\mathbf{p}}} \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|)\, I_{\mathbf{q}},$$

where normalization factor $W_{\mathbf{p}}$ ensures pixel weights sum to 1.0:

$$W_{\mathbf{p}} = \sum_{\mathbf{q} \in \mathcal{S}} G_{\sigma_s}(\|\mathbf{p} - \mathbf{q}\|)\, G_{\sigma_r}(|I_{\mathbf{p}} - I_{\mathbf{q}}|).$$



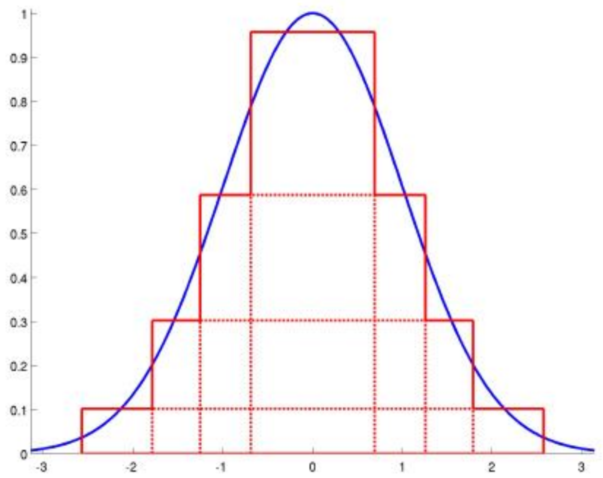
Bilateral filtering

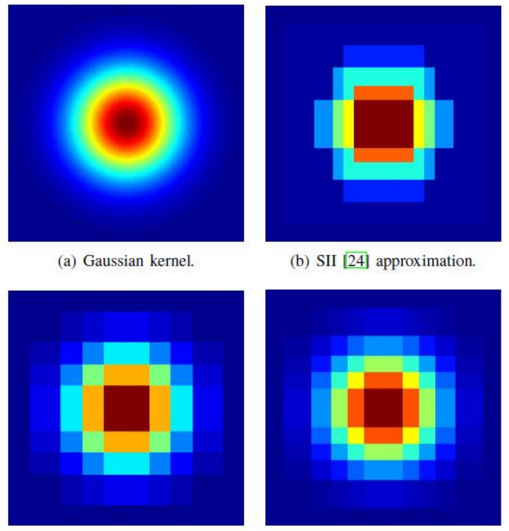
Bilateral Filtering with CUDA

Lasse Kløjgaard Staal, 20072300*

Efficient and Accurate Gaussian Image Filtering Using Running Sums

Elhanan Elboher and Michael Werman





(a) Gaussian kernel.

(b) SII [24] approximation.

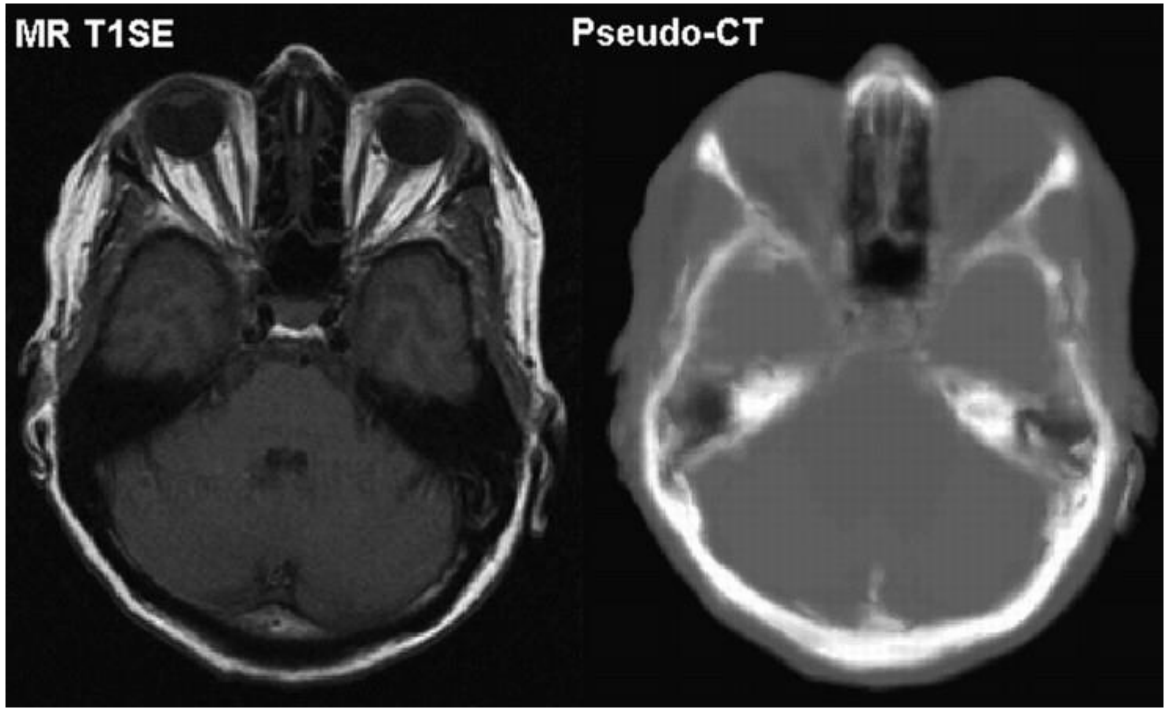
(c) Proposed approximation (4 constants).

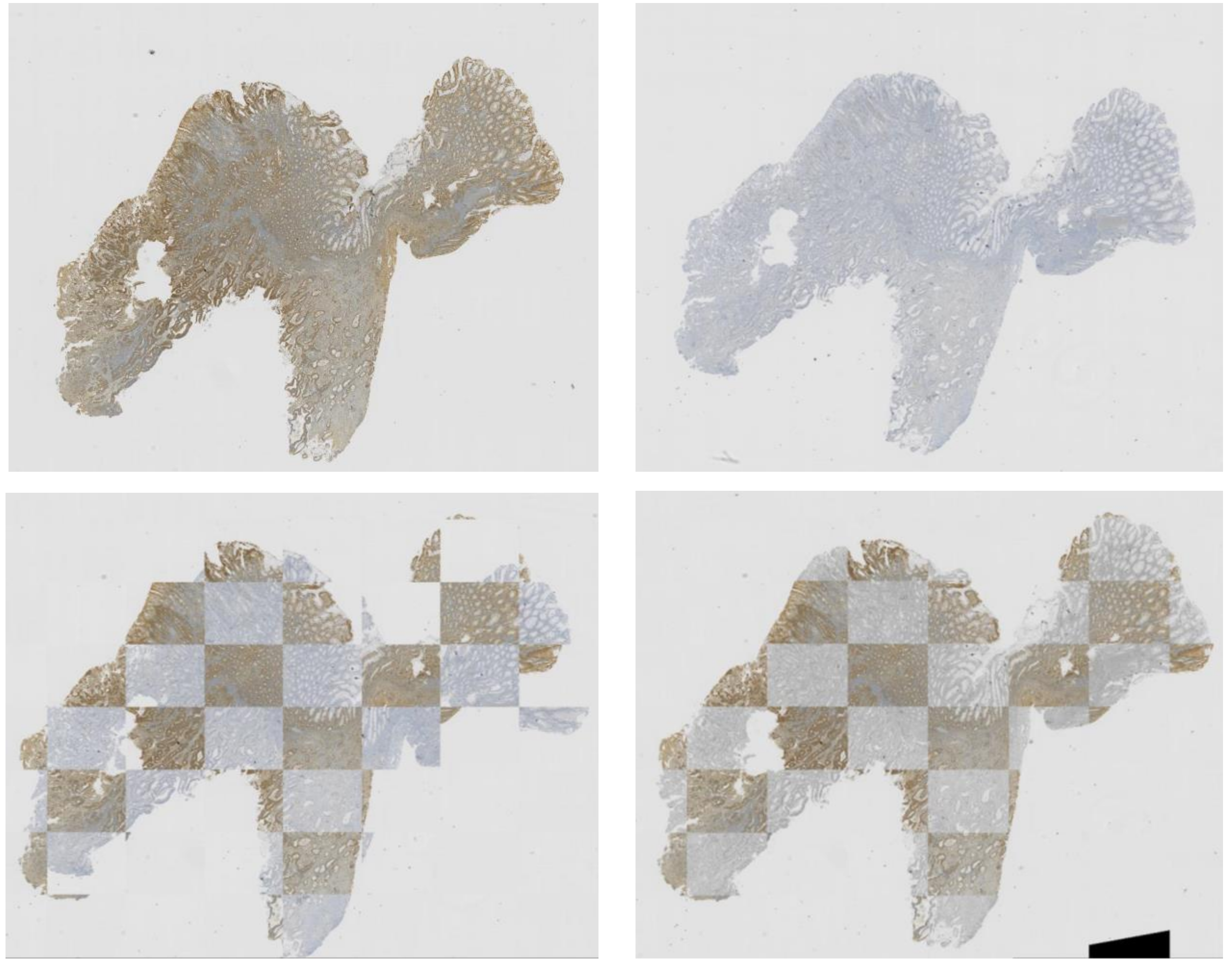
(d) Proposed approximation (5 constants).

```
__global__
void bilateralFilterGPU_v5(float3* output, uint2 dims, int
radius, float* kernel, float variance, float sqrt_pi_variance)
{
   const unsigned int idx = blockIdx.x*blockDim.x + threadIdx.x;

   uint2 pos = idx_to_co(idx,dims);

   if(pos.x >= dims.x || pos.y >= dims.y) return;

   float3 currentColor = make_float3(tex1Dfetch(tex,
3*idx),tex1Dfetch(tex, 3*idx+1),tex1Dfetch(tex, 3*idx+2));

   float3 res = make_float3(0.0f,0.0f,0.0f);
   float3 normalization = make_float3(0.0f,0.0f,0.0f);
   float3 weight;
   for(int i = -radius; i <= radius; i++) {
      for(int j = -radius; j <= radius; j++) {

         int x_sample = pos.x+i;
         int y_sample = pos.y+j;

         //mirror edges
         if( x_sample < 0) x_sample = -x_sample;
         if( y_sample < 0) y_sample = -y_sample;
         if( x_sample > dims.x - 1) x_sample = dims.x - 1 - i;
         if( y_sample > dims.y - 1) y_sample = dims.y - 1 - j;

         int tempPos =
co_to_idx(make_uint2(x_sample,y_sample),dims);

         float3 tmpColor = make_float3(tex1Dfetch(tex,
3*tempPos),tex1Dfetch(tex, 3*tempPos+1),tex1Dfetch(tex,
3*tempPos+2));//input[tempPos];

         float gauss_spatial =
kernel[co_to_idx(make_uint2(i+radius,j+radius),make_uint2(radius*
2+1,radius*2+1))];

         weight.x = gauss_spatial *
gaussian1d_gpu_reg((currentColor.x -
tmpColor.x),variance,sqrt_pi_variance);
         weight.y = gauss_spatial *
gaussian1d_gpu_reg((currentColor.y -
tmpColor.y),variance,sqrt_pi_variance);
         weight.z = gauss_spatial *
gaussian1d_gpu_reg((currentColor.z -
tmpColor.z),variance,sqrt_pi_variance);

         normalization = normalization + weight;
         res = res + (tmpColor * weight);

      }
   }

   res.x /= normalization.x;
   res.y /= normalization.y;
   res.z /= normalization.z;
   output[idx] = res;
}
```

```
__global__
void bilateralFilterGPU_v6(float3* output, uint2 dims, int
radius, float* kernel, float variance, float sqrt_pi_variance)
{
   const unsigned int idx = blockIdx.x*blockDim.x + threadIdx.x;

   uint2 pos = idx_to_co(idx,dims);

   if(pos.x >= dims.x || pos.y >= dims.y) return;

   float3 currentColor = make_float3(tex1Dfetch(tex_red,
idx),tex1Dfetch(tex_green, idx),tex1Dfetch(tex_blue, idx));

   float3 res = make_float3(0.0f,0.0f,0.0f);
   float3 normalization = make_float3(0.0f,0.0f,0.0f);

   for(int i = -radius; i <= radius; i++) {
      for(int j = -radius; j <= radius; j++) {

         int x_sample = pos.x+i;
         int y_sample = pos.y+j;

         //mirror edges
         if( x_sample < 0) x_sample = -x_sample;
         if( y_sample < 0) y_sample = -y_sample;
         if( x_sample > dims.x - 1) x_sample = dims.x - 1 - i;
         if( y_sample > dims.y - 1) y_sample = dims.y - 1 - j;

         int tempPos =
co_to_idx(make_uint2(x_sample,y_sample),dims);

         float3 tmpColor = make_float3(tex1Dfetch(tex_red,
tempPos),tex1Dfetch(tex_green, tempPos),tex1Dfetch(tex_blue,
tempPos));

         float gauss_spatial =
kernel[co_to_idx(make_uint2(i+radius,j+radius),make_uint2(radius*
2+1,radius*2+1))];

         float3 gauss_range;
         gauss_range.x = gaussian1d_gpu_reg(currentColor.x -
tmpColor.x, variance, sqrt_pi_variance);
         gauss_range.y = gaussian1d_gpu_reg(currentColor.y -
tmpColor.y, variance, sqrt_pi_variance);
         gauss_range.z = gaussian1d_gpu_reg(currentColor.z -
tmpColor.z, variance, sqrt_pi_variance);

         float3 weight = gauss_spatial * gauss_range;

         normalization = normalization + weight;
         res = res + (tmpColor * weight);

      }
   }

   res.x /= normalization.x;
   res.y /= normalization.y;
   res.z /= normalization.z;
   output[idx] = res;
}
```

Mutual Information image registration (1-2 students)





CUDA code available on http://users.cecs.anu.edu.au/~ramtin/cuda.htm

http://users.cecs.anu.edu.au/~ramtin/papers/2007/DICTA_2007a.pdf

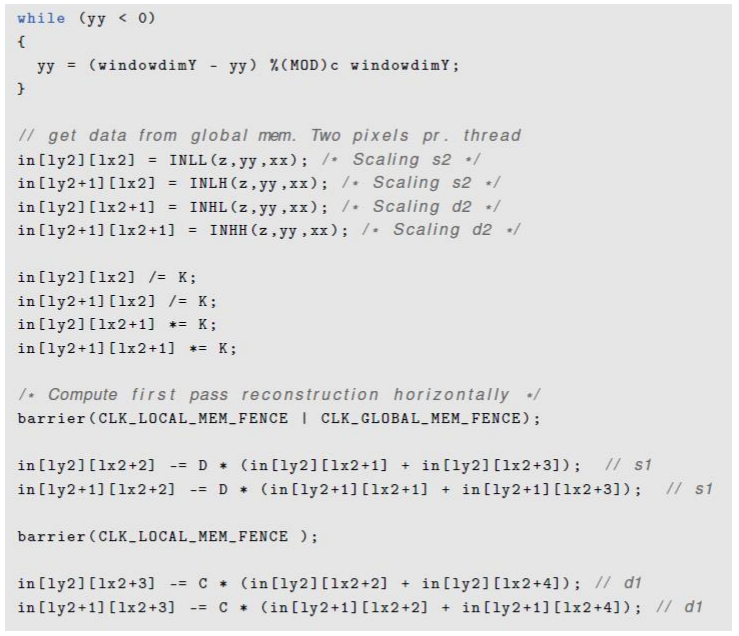
https://lirias.kuleuven.be/bitstream/123456789/28116/1/Maes97TMI.pdf

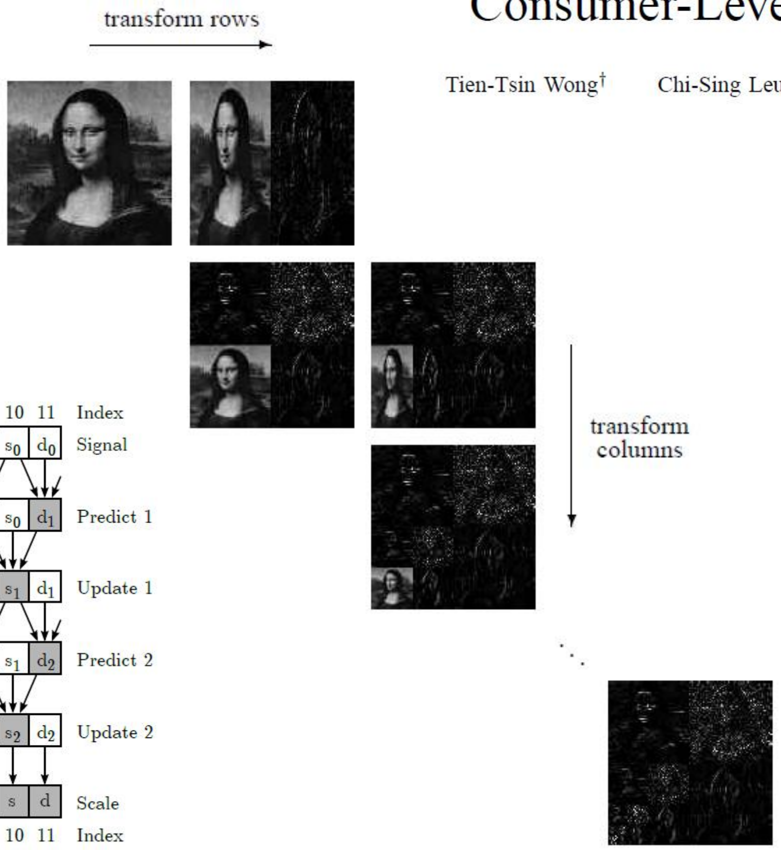
2D Wavelet Transform

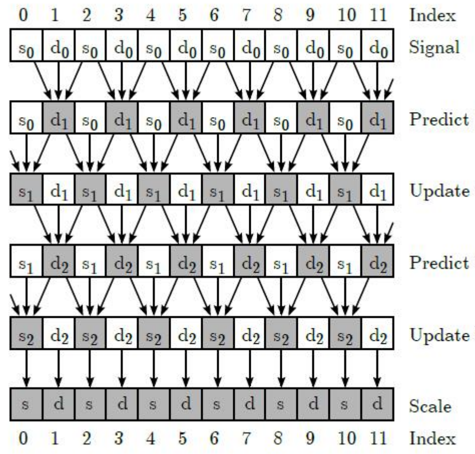
Wavelet transform for video coders using GPU

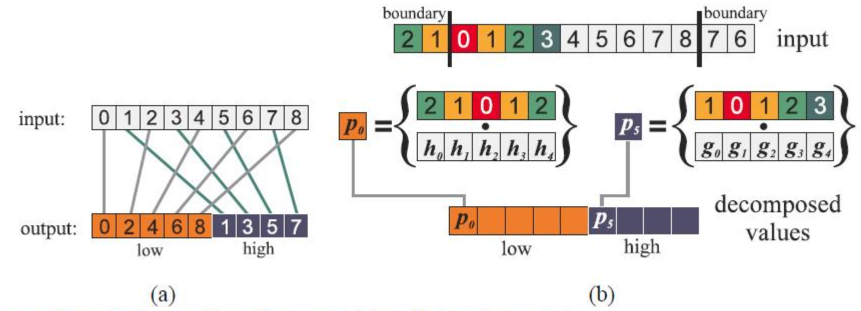
Discrete Wavelet Transform on Consumer-Level Graphics Hardware

Tien-Tsin Wong[†] Chi-Sing Leung[‡] Pheng-Ann Heng[†] Jianqing Wang[†]


```
while (yy < 0)
{
  yy = (windowdimY - yy) %(MOD)c windowdimY;
}

// get data from global mem. Two pixels pr. thread
in[ly2][lx2] = INLL(z,yy,xx); /* Scaling s2 */
in[ly2+1][lx2] = INLH(z,yy,xx); /* Scaling s2 */
in[ly2][lx2+1] = INHL(z,yy,xx); /* Scaling d2 */
in[ly2+1][lx2+1] = INHH(z,yy,xx); /* Scaling d2 */

in[ly2][lx2] /= K;
in[ly2+1][lx2] /= K;
in[ly2][lx2+1] *= K;
in[ly2+1][lx2+1] *= K;

/* Compute first pass reconstruction horizontally */
barrier(CLK_LOCAL_MEM_FENCE | CLK_GLOBAL_MEM_FENCE);

in[ly2][lx2+2] -= D * (in[ly2][lx2+1] + in[ly2][lx2+3]);  // s1
in[ly2+1][lx2+2] -= D * (in[ly2+1][lx2+1] + in[ly2+1][lx2+3]);  // s1

barrier(CLK_LOCAL_MEM_FENCE );

in[ly2][lx2+3] -= C * (in[ly2][lx2+2] + in[ly2][lx2+4]); // d1
in[ly2+1][lx2+3] -= C * (in[ly2+1][lx2+2] + in[ly2+1][lx2+4]); // d1
```







(a) Mapping to the base positions. (b) Decomposition with boundary extension.

Volumetric Image & Audio processing

Sound wave propagation

Jukka Saarelma

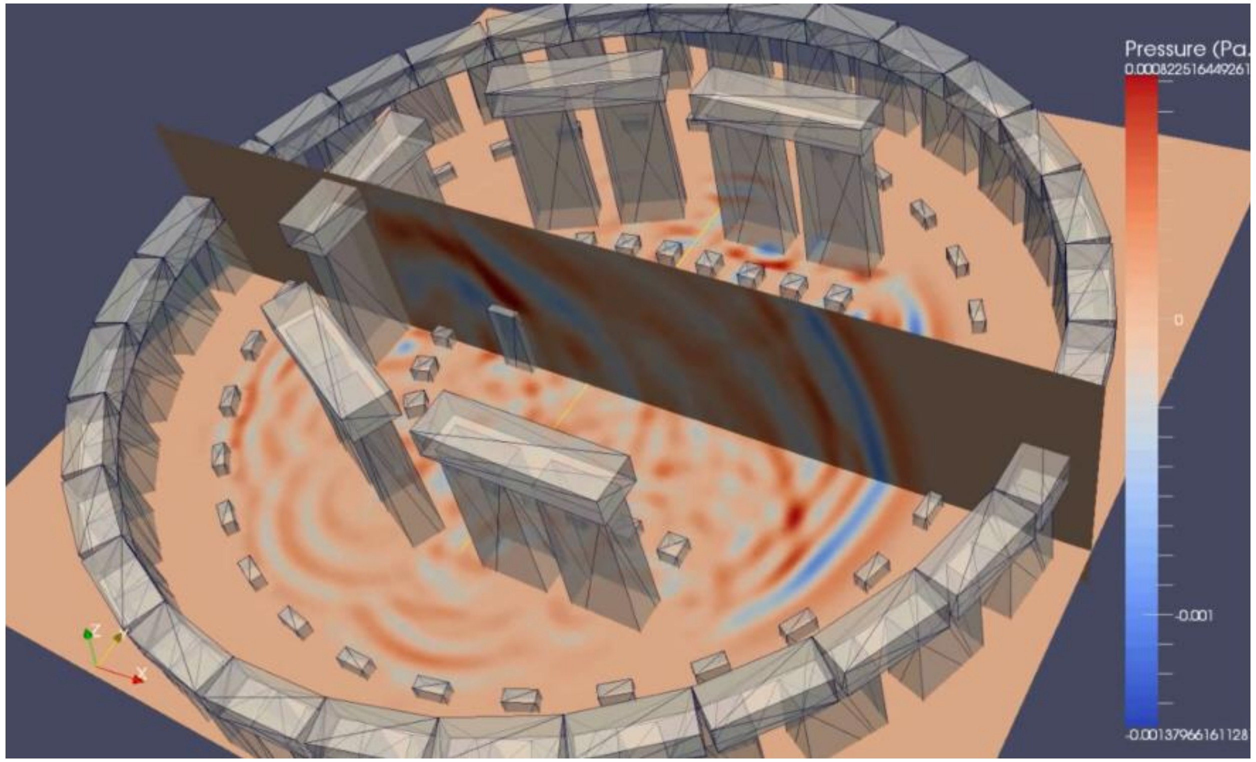
Finite-difference time-domain solver for room acoustics using graphics processing units

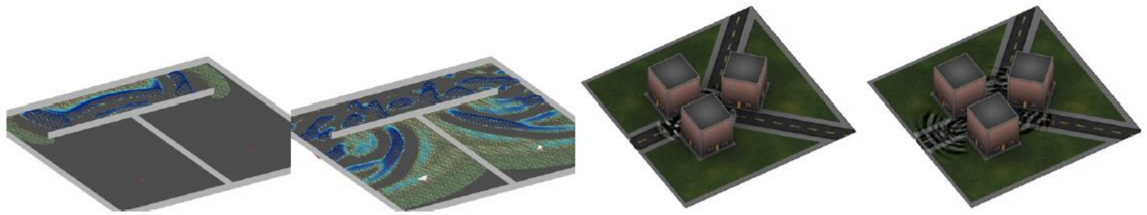
$$c^2 \nabla^2 p = \frac{\partial^2 p}{\partial^2 t},$$

$$c^2 = \kappa \frac{p_0}{\rho_0}.$$

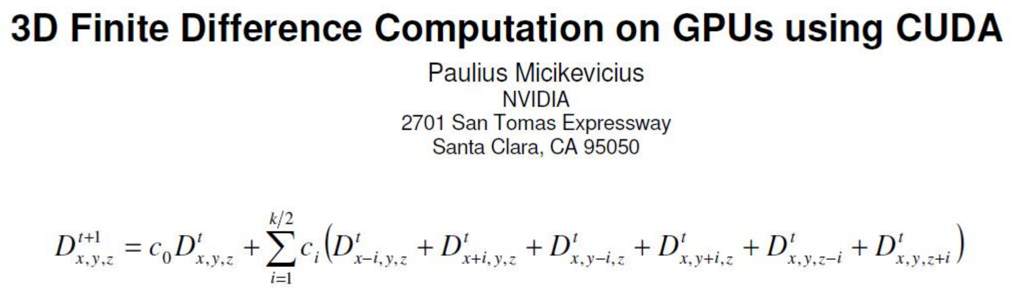
Sound Wave Propagation Applied in Games

Marcelo Zamith, Erick Passos, Diego Brandão,
Anselmo Montenegro, Esteban Clua, Mauricio Kischinhevsky, Regina C.P. Leal-Toledo





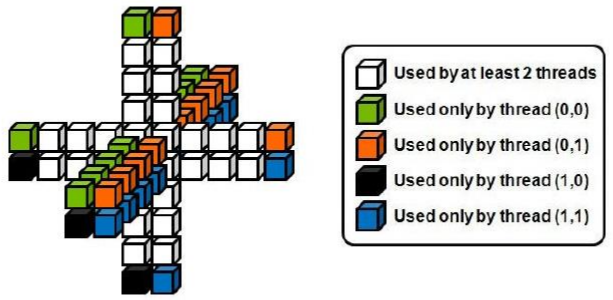
Finite Difference Time Domain

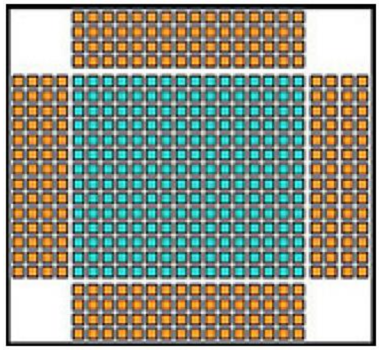


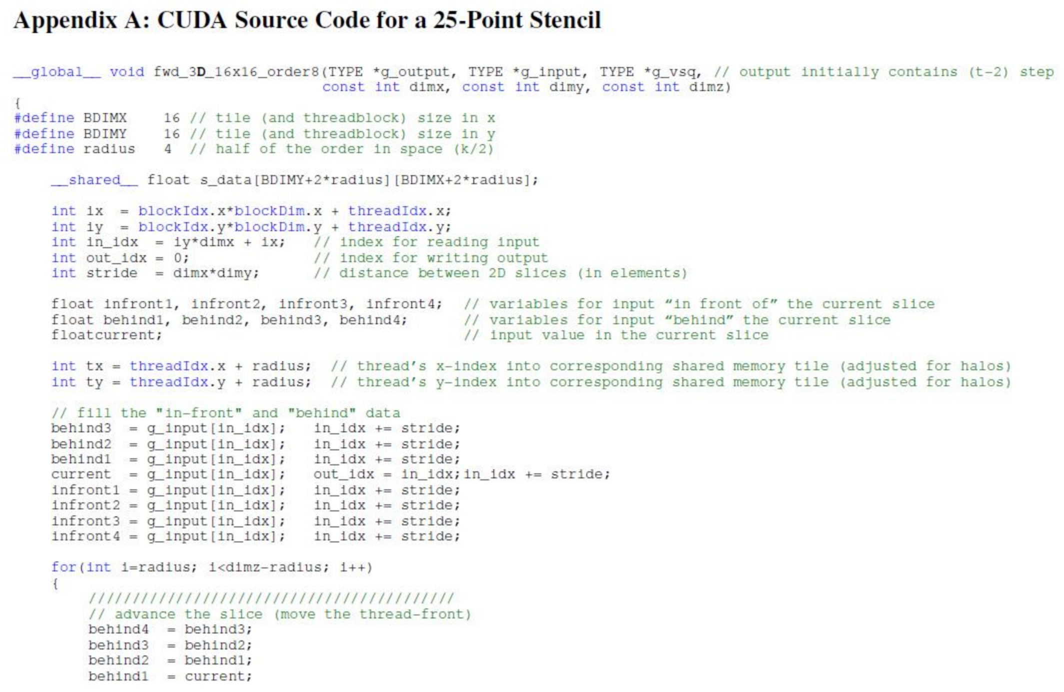
3D Finite Difference Computation on GPUs using CUDA

Paulius Micikevicius
NVIDIA
2701 San Tomas Expressway
Santa Clara, CA 95050

$$D^{t+1}_{x,y,z} = c_0 D^t_{x,y,z} + \sum_{i=1}^{k/2} c_i \left(D^t_{x-i,y,z} + D^t_{x+i,y,z} + D^t_{x,y-i,z} + D^t_{x,y+i,z} + D^t_{x,y,z-i} + D^t_{x,y,z+i} \right)$$


Used by at least 2 threads
Used only by thread (0,0)
Used only by thread (0,1)
Used only by thread (1,0)
Used only by thread (1,1)





Appendix A: CUDA Source Code for a 25-Point Stencil

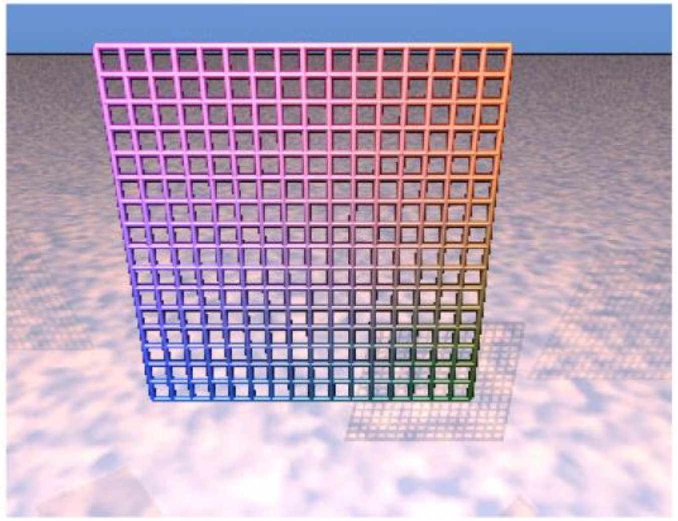
```
__global__ void fwd_3D_16x16_order8(TYPE *g_output, TYPE *g_input, TYPE *g_vsq, // output initially contains (t-2) step
                                    const int dimx, const int dimy, const int dimz)
{
#define BDIMX    16 // tile (and threadblock) size in x
#define BDIMY    16 // tile (and threadblock) size in y
#define radius   4  // half of the order in space (k/2)

    __shared__ float s_data[BDIMY+2*radius][BDIMX+2*radius];

    int ix  = blockIdx.x*blockDim.x + threadIdx.x;
    int iy  = blockIdx.y*blockDim.y + threadIdx.y;
    int in_idx  = iy*dimx + ix;   // index for reading input
    int out_idx = 0;              // index for writing output
    int stride  = dimx*dimy;      // distance between 2D slices (in elements)

    float infront1, infront2, infront3, infront4;  // variables for input “in front of” the current slice
    float behind1, behind2, behind3, behind4;      // variables for input “behind” the current slice
    floatcurrent;                                  // input value in the current slice

    int tx = threadIdx.x + radius;  // thread’s x-index into corresponding shared memory tile (adjusted for halos)
    int ty = threadIdx.y + radius;  // thread’s y-index into corresponding shared memory tile (adjusted for halos)

    // fill the "in-front" and "behind" data
    behind3  = g_input[in_idx];   in_idx += stride;
    behind2  = g_input[in_idx];   in_idx += stride;
    behind1  = g_input[in_idx];   in_idx += stride;
    current  = g_input[in_idx];   out_idx = in_idx;in_idx += stride;
    infront1 = g_input[in_idx];   in_idx += stride;
    infront2 = g_input[in_idx];   in_idx += stride;
    infront3 = g_input[in_idx];   in_idx += stride;
    infront4 = g_input[in_idx];   in_idx += stride;

    for(int i=radius; i<dimz-radius; i++)
    {
        //////////////////////////////////////////
        // advance the slice (move the thread-front)
        behind4  = behind3;
        behind3  = behind2;
        behind2  = behind1;
        behind1  = current;
```

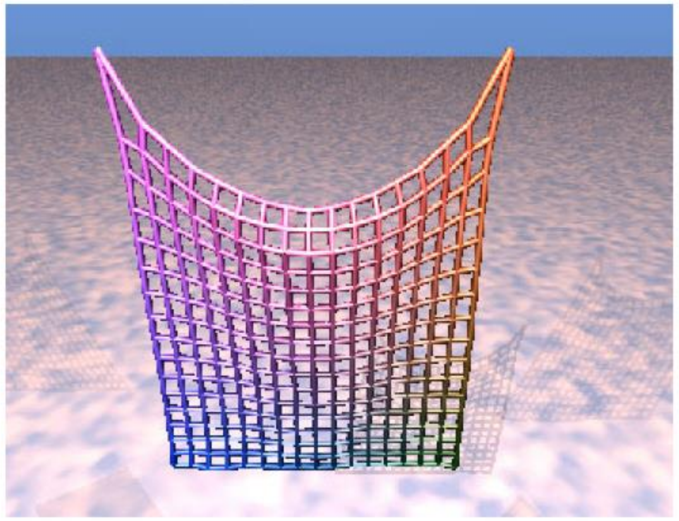
3D graphics processing

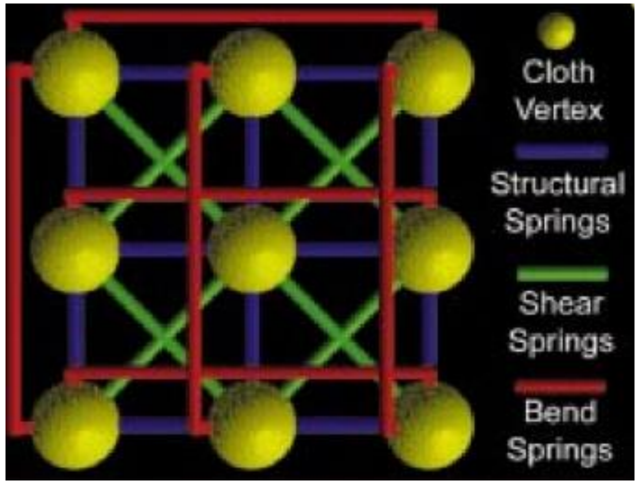
Cloth simulation

Deformation Constraints in a Mass-Spring Model
to Describe Rigid Cloth Behavior

Xavier Provot



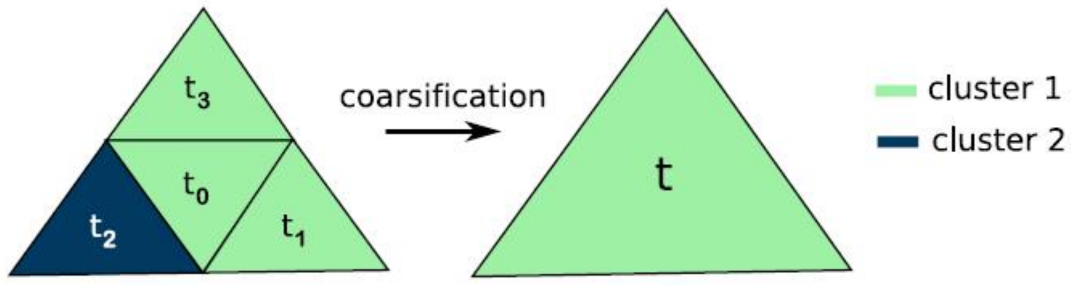


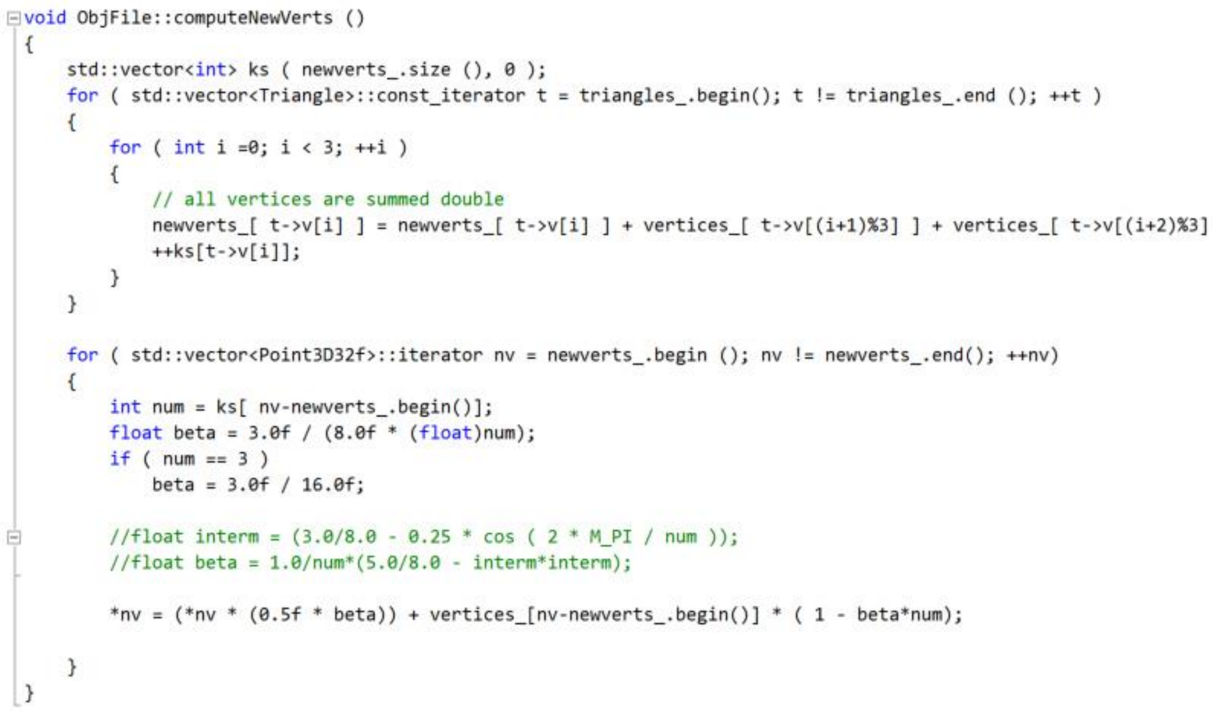


3D wavelet transform

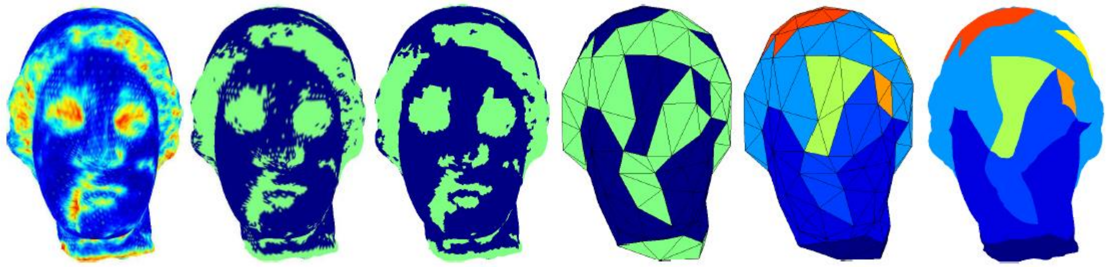
Adapted Semi-Regular 3-D Mesh Coding Based on a Wavelet Segmentation

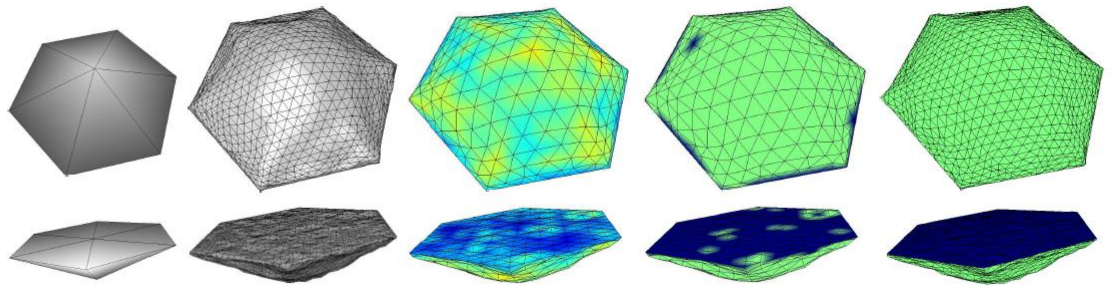
Céline Roudet[a], Florent Dupont[a] and Atilla Baskurt[b]





```
void ObjFile::computeNewVerts ()
{
    std::vector<int> ks ( newverts_.size (), 0 );
    for ( std::vector<Triangle>::const_iterator t = triangles_.begin(); t != triangles_.end (); ++t )
    {
        for ( int i =0; i < 3; ++i )
        {
            // all vertices are summed double
            newverts_[ t->v[i] ] = newverts_[ t->v[i] ] + vertices_[ t->v[(i+1)%3] ] + vertices_[ t->v[(i+2)%3]
            ++ks[t->v[i]];
        }
    }

    for ( std::vector<Point3D32f>::iterator nv = newverts_.begin (); nv != newverts_.end(); ++nv)
    {
        int num = ks[ nv-newverts_.begin()];
        float beta = 3.0f / (8.0f * (float)num);
        if ( num == 3 )
            beta = 3.0f / 16.0f;

        //float interm = (3.0/8.0 - 0.25 * cos ( 2 * M_PI / num ));
        //float beta = 1.0/num*(5.0/8.0 - interm*interm);

        *nv = (*nv * (0.5f * beta)) + vertices_[nv-newverts_.begin()] * ( 1 - beta*num);

    }
}
```





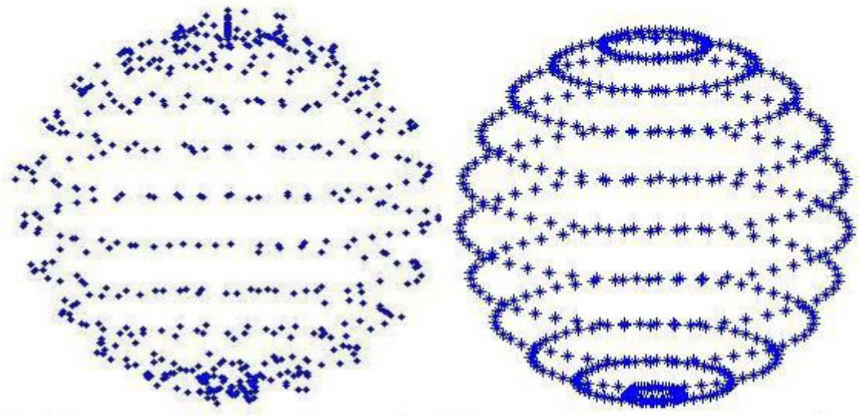
Least squares object fitting

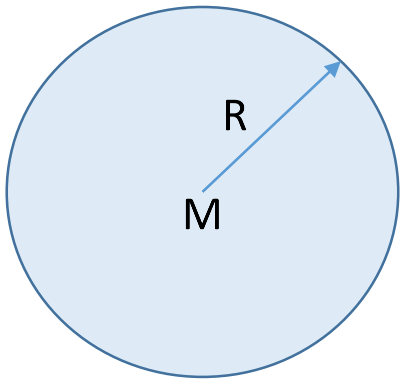
Clemson University
TigerPrints

All Theses

12-2007

Least Squares Fitting of Analytic Primitives on a GPU





$$F\ x_0,y_0,z_0,r_0\ = \sum \sqrt{x_i-x_0\ ^2 + \ y_i-y_0\ ^2 + \ z_i-z_0\ ^2} - r_0\ ^2 \qquad (4.34)$$

The partial derivatives of the function f with respect to each of the parameters x_0, y_0, z_0, r_0 are given by

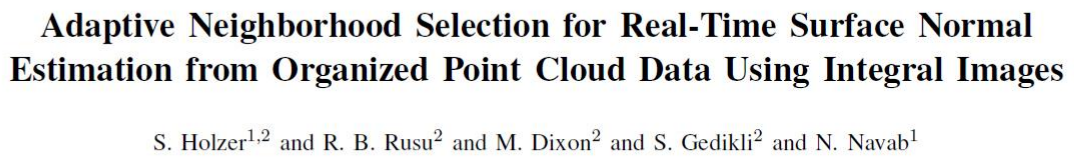
$$\frac{\partial f}{\partial x_0} = -(x_i - x_0)/\ \ f + r_0 \qquad (4.35)$$

$$\frac{\partial f}{\partial y_0} = -(y_i - y_0)/\ \ f + r_0 \qquad (4.36)$$

$$\frac{\partial f}{\partial z_0} = -(z_i - z_0)/\ \ f + r_0 \qquad (4.37)$$

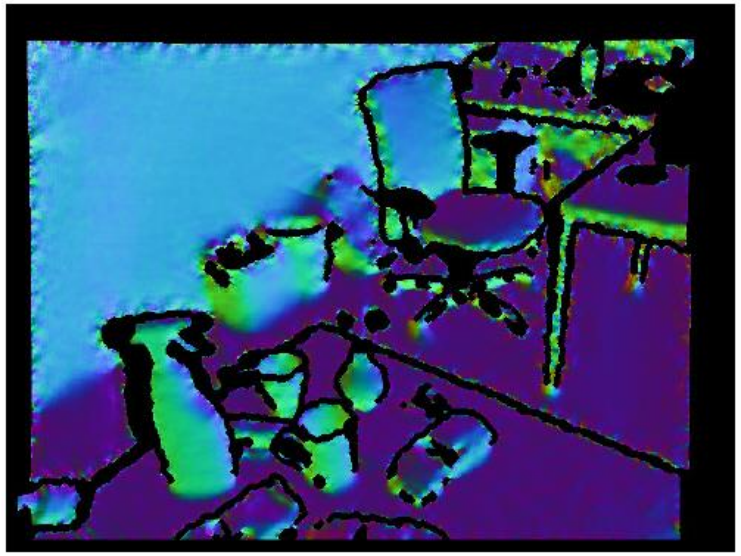
$$\frac{\partial f}{\partial r_0} = -1 \qquad (4.38)$$

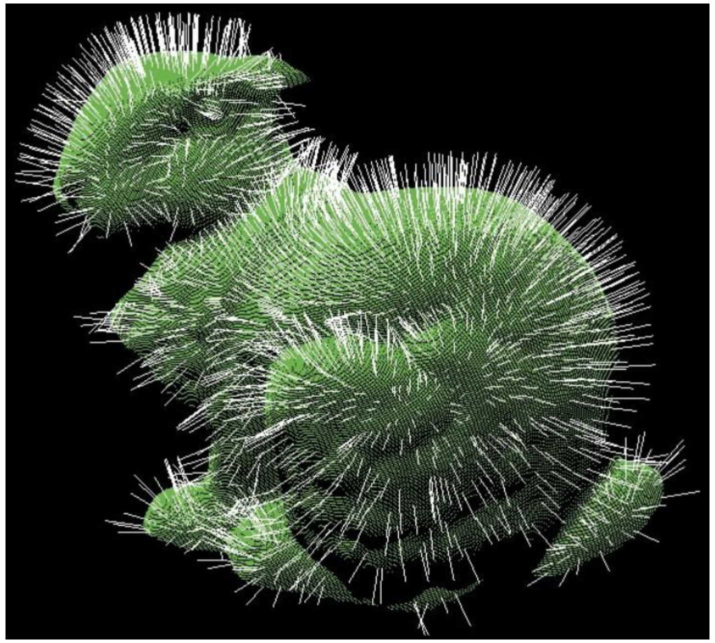
Surface normal from depth images

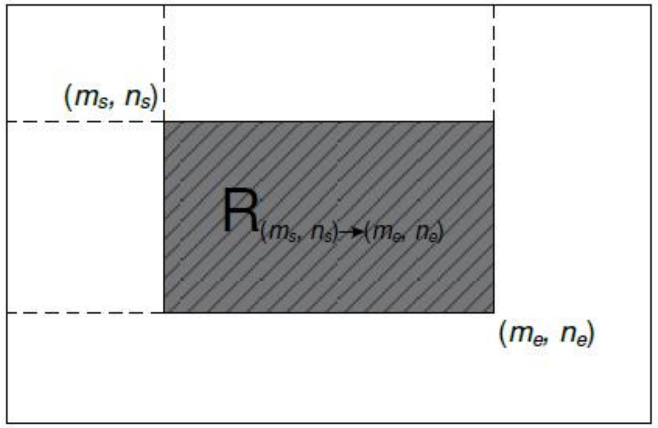


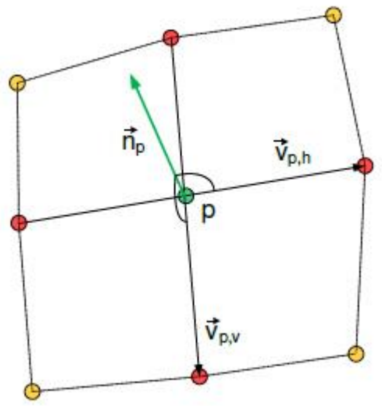
Adaptive Neighborhood Selection for Real-Time Surface Normal Estimation from Organized Point Cloud Data Using Integral Images

S. Holzer[1,2] and R. B. Rusu[2] and M. Dixon[2] and S. Gedikli[2] and N. Navab[1]

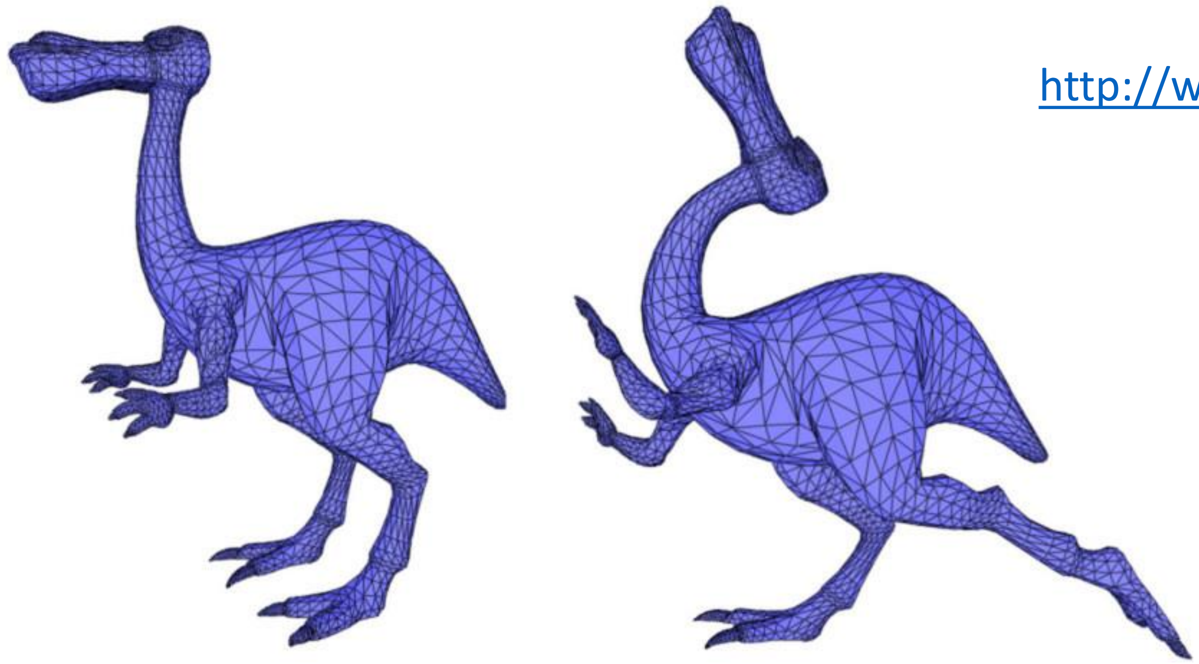


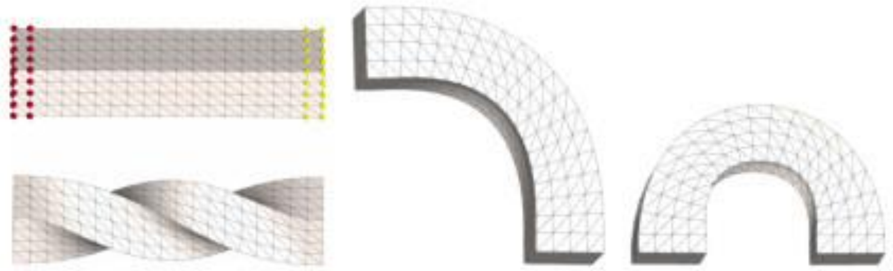






As-Rigid-As-Possible Surface Deformations (1-2 students)





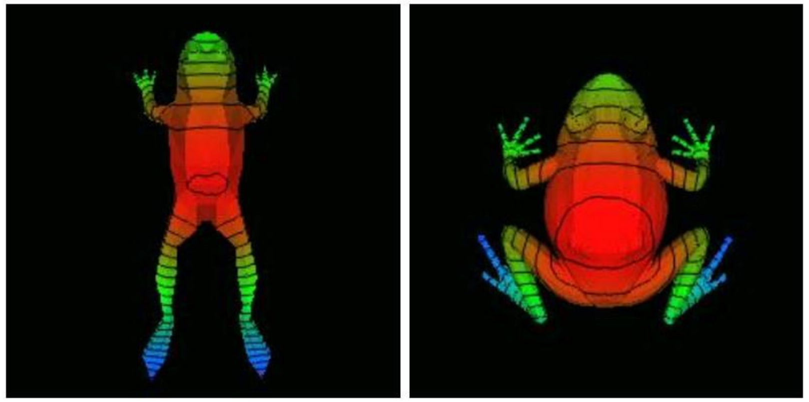
http://www.igl.ethz.ch/projects/ARAP/arap_web.pdf

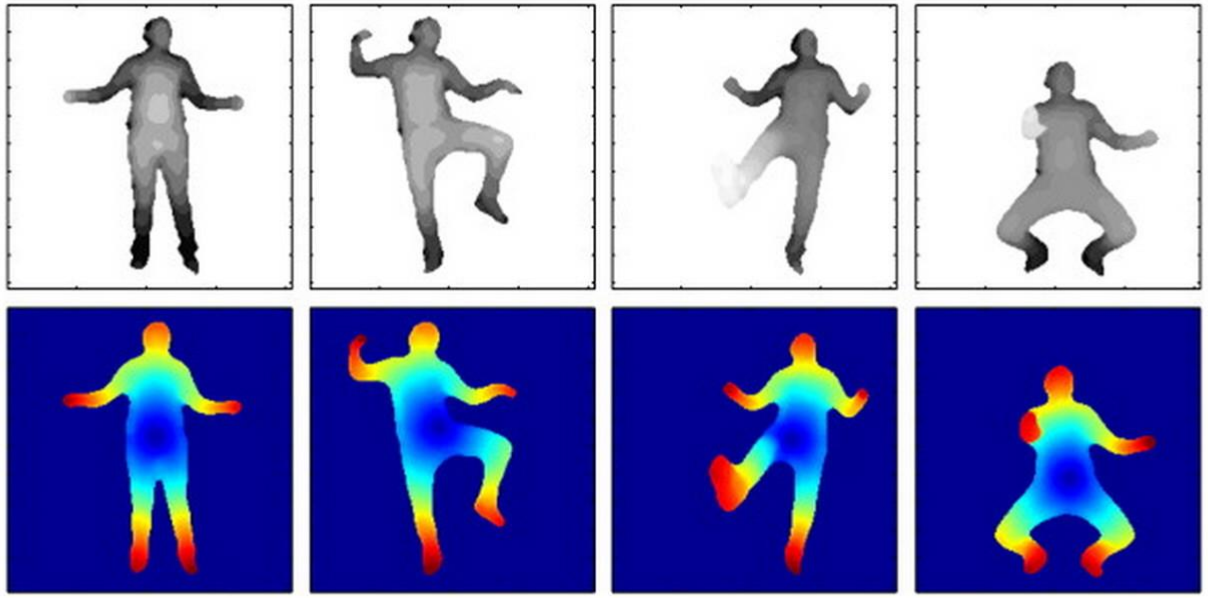
$$E(\mathcal{C}_i,\mathcal{C}_i') = \sum_{j\in\mathcal{N}(i)} w_{ij} \left\| (\mathbf{p}_i' - \mathbf{p}_j') - \mathbf{R}_i(\mathbf{p}_i - \mathbf{p}_j) \right\|^2$$

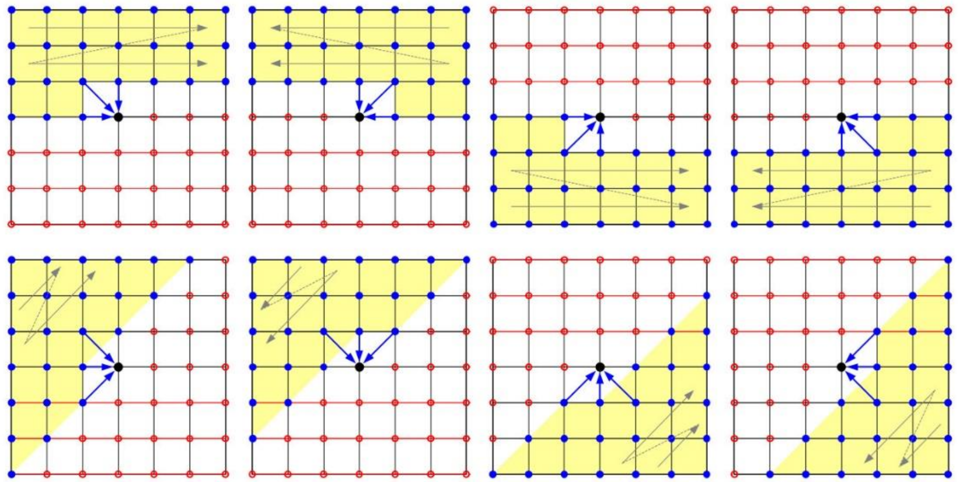
$$\sum_j w_{ij} \left(\mathbf{e}_{ij}' - \mathbf{R}_i \mathbf{e}_{ij}\right)^T \left(\mathbf{e}_{ij}' - \mathbf{R}_i \mathbf{e}_{ij}\right) =$$
$$= \sum_j w_{ij} \left(\mathbf{e}_{ij}'^T \mathbf{e}_{ij}' - 2\mathbf{e}_{ij}'^T \mathbf{R}_i \mathbf{e}_{ij} + \mathbf{e}_{ij}^T \mathbf{R}_i^T \mathbf{R}_i \mathbf{e}_{ij}\right) =$$
$$= \sum_j w_{ij} \left(\mathbf{e}_{ij}'^T \mathbf{e}_{ij}' - 2\mathbf{e}_{ij}'^T \mathbf{R}_i \mathbf{e}_{ij} + \mathbf{e}_{ij}^T \mathbf{e}_{ij}\right).$$

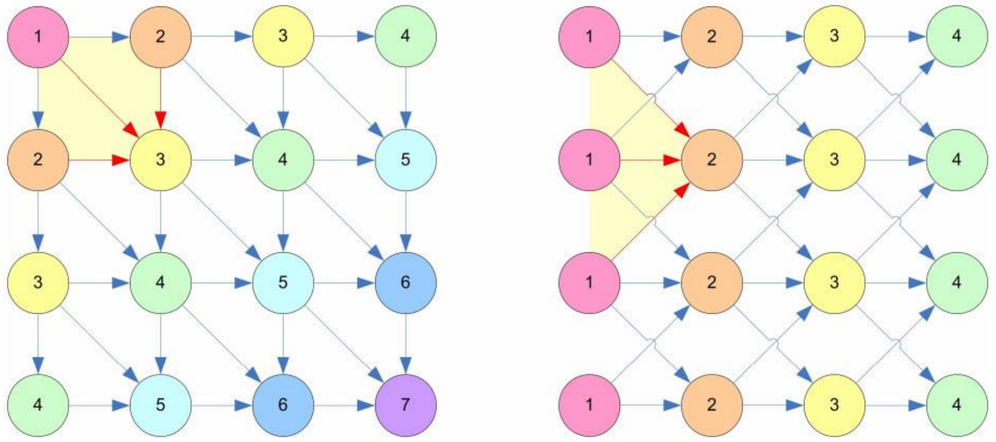
C++ source code available
SVD minimization with CUDA library

Calculation of Geodesic distances (1-2 students)



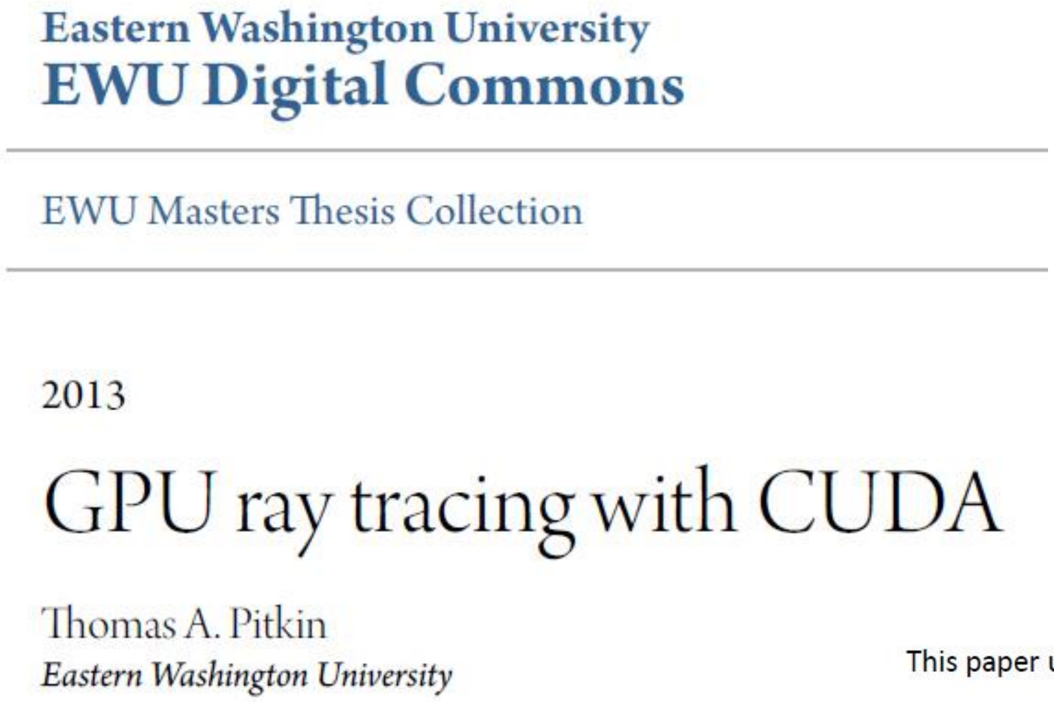






http://vista.eng.tau.ac.il/publications/WebDevBroBroKimTOG08.pdf

GPU ray tracer (1-2 students)



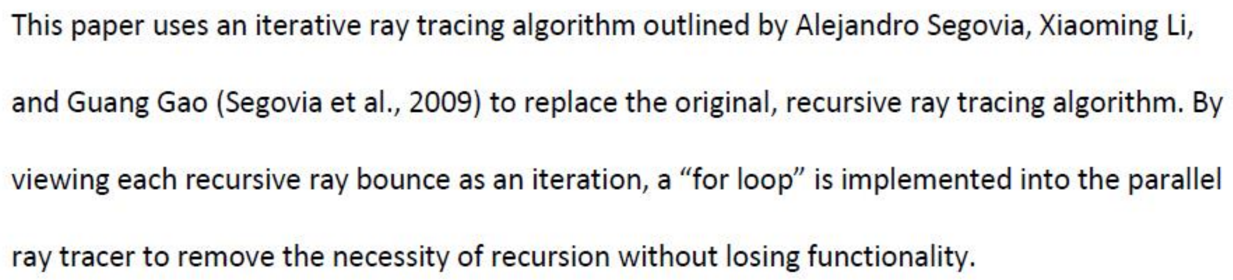
Eastern Washington University
EWU Digital Commons

EWU Masters Thesis Collection

2013

GPU ray tracing with CUDA

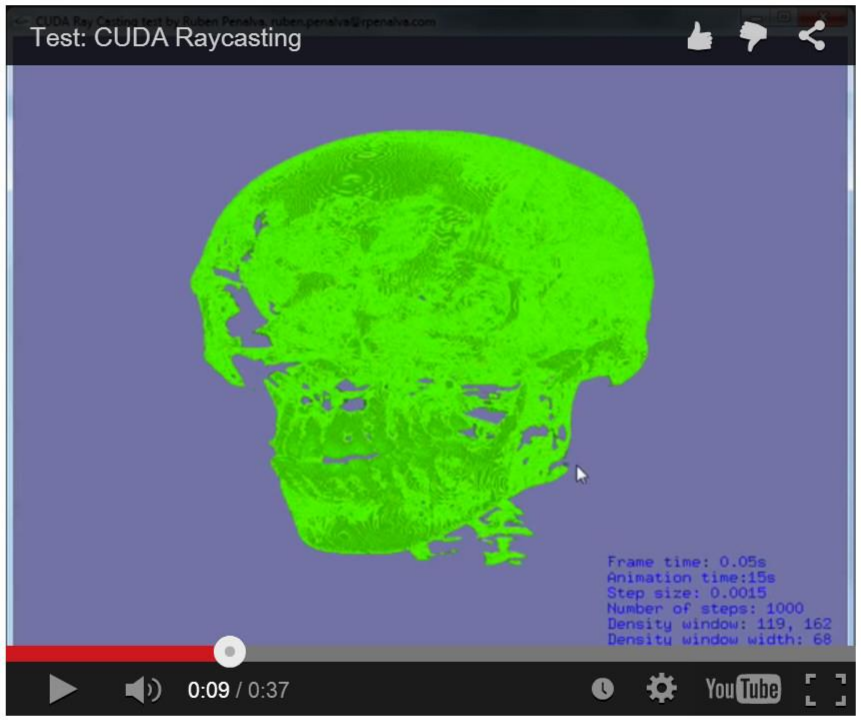
Thomas A. Pitkin
Eastern Washington University



This paper uses an iterative ray tracing algorithm outlined by Alejandro Segovia, Xiaoming Li, and Guang Gao (Segovia et al., 2009) to replace the original, recursive ray tracing algorithm. By viewing each recursive ray bounce as an iteration, a "for loop" is implemented into the parallel ray tracer to remove the necessity of recursion without losing functionality.

http://www.kevinbeason.com/smallpt/

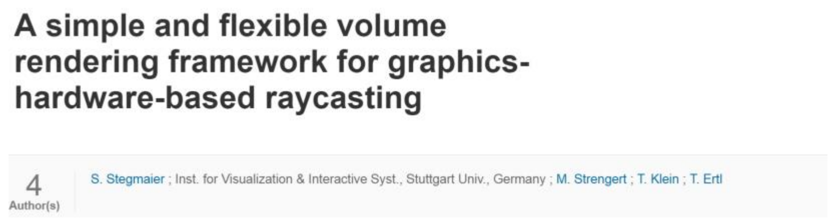
CUDA volume raytracer


Test: CUDA Raycasting

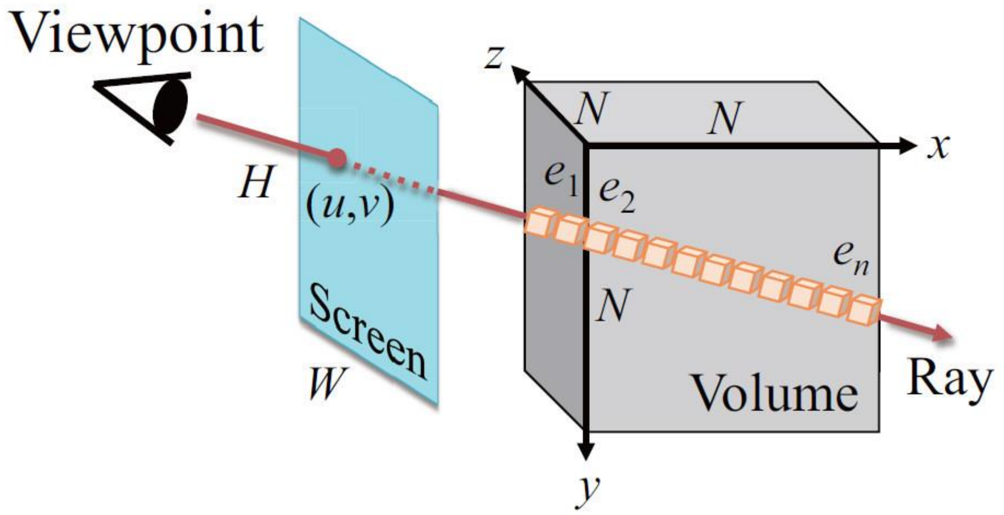
```
Frame time: 0.05s
Animation time:15s
Step size: 0.0015
Number of steps: 1000
Density window: 119, 162
Density window width: 68
```

http://www.rpenalva.com/blog/?p=229

CUDA code available, but needs adaptations and profiling


A simple and flexible volume rendering framework for graphics-hardware-based raycasting

4 Author(s) | S. Stegmaier ; Inst. for Visualization & Interactive Syst., Stuttgart Univ., Germany ; M. Strengert ; T. Klein ; T. Ertl


Viewpoint
H
(u,v)
Screen
W
z
N
N
x
e_1
e_2
e_n
N
Volume
Ray
y