GPGPU programming with image processing applications

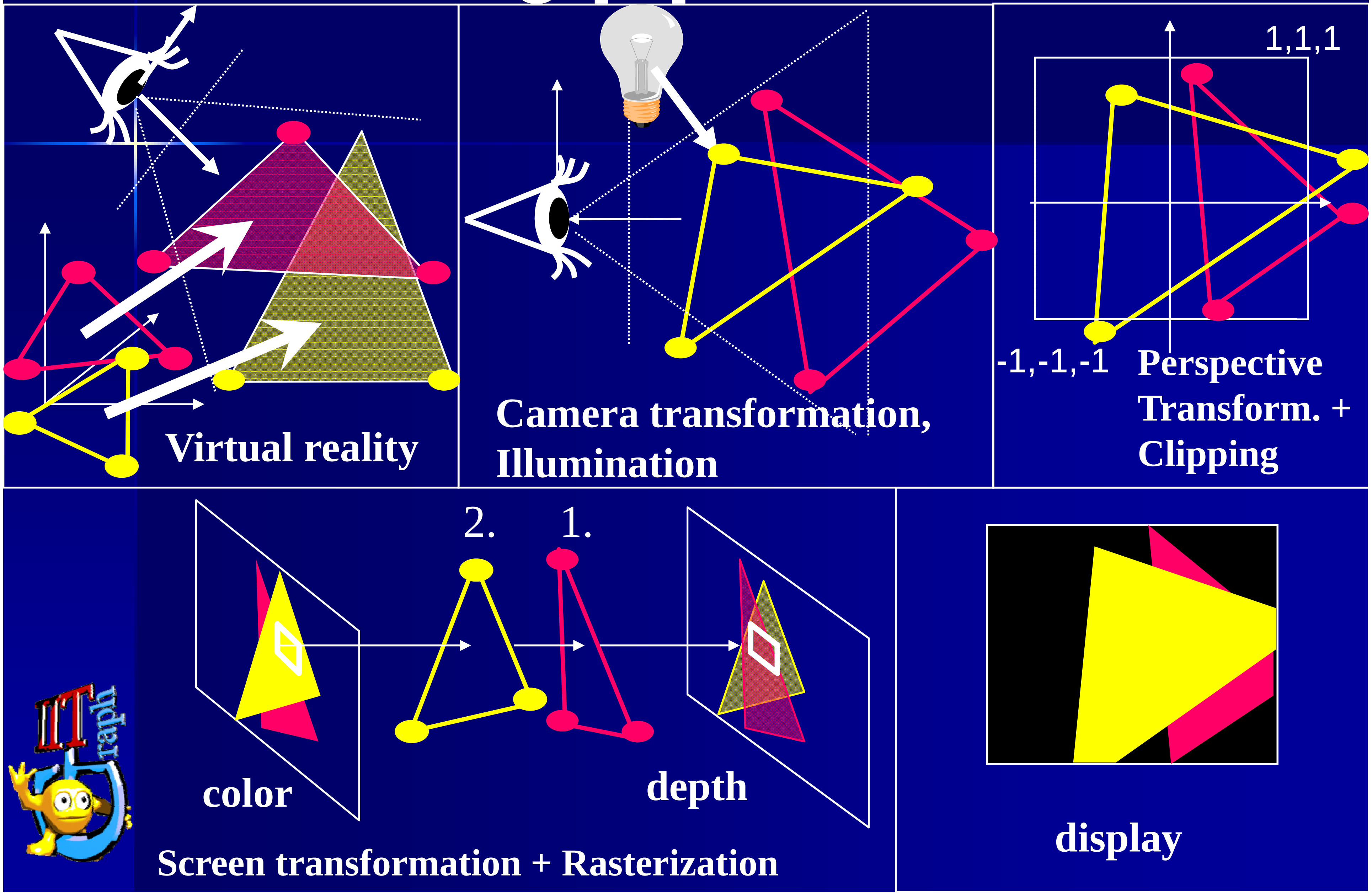
Szirmay-Kalos László

SSIP 2011

Agenda

- Incremental rendering pipeline
- GPU and its programming models:
- Shader API (Shader Model 3, Cg)
  - Filtering
  - Image distortions
  - Global image functions (average)
  - Histogram
- Gather or Scatter
- CUDA
  - Matrix operations
  - Fluid dynamics
  - N-body (molecular dynamics)

Rendering pipeline



Virtual reality

Camera transformation, Illumination

Perspective Transform. + Clipping

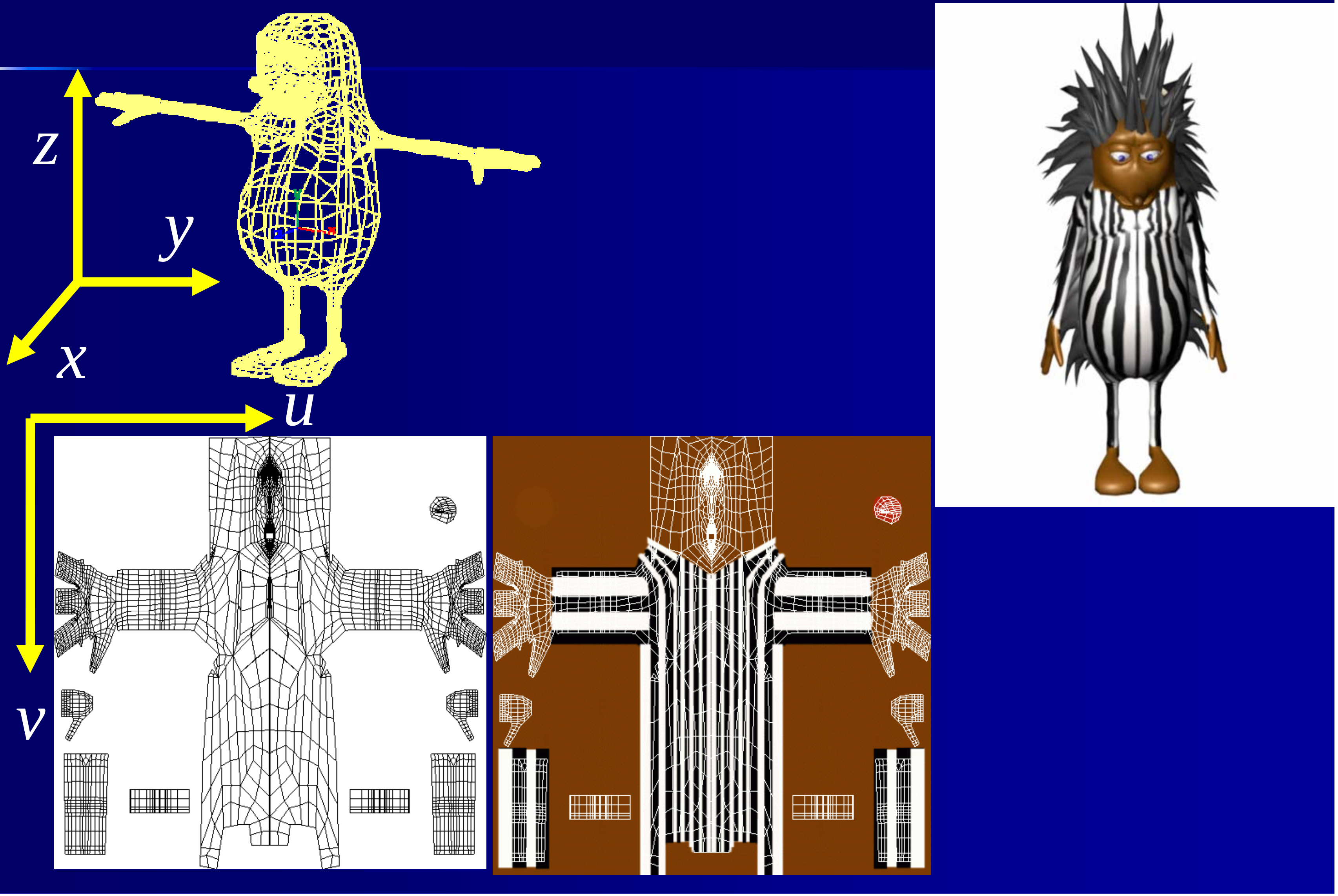
color

depth

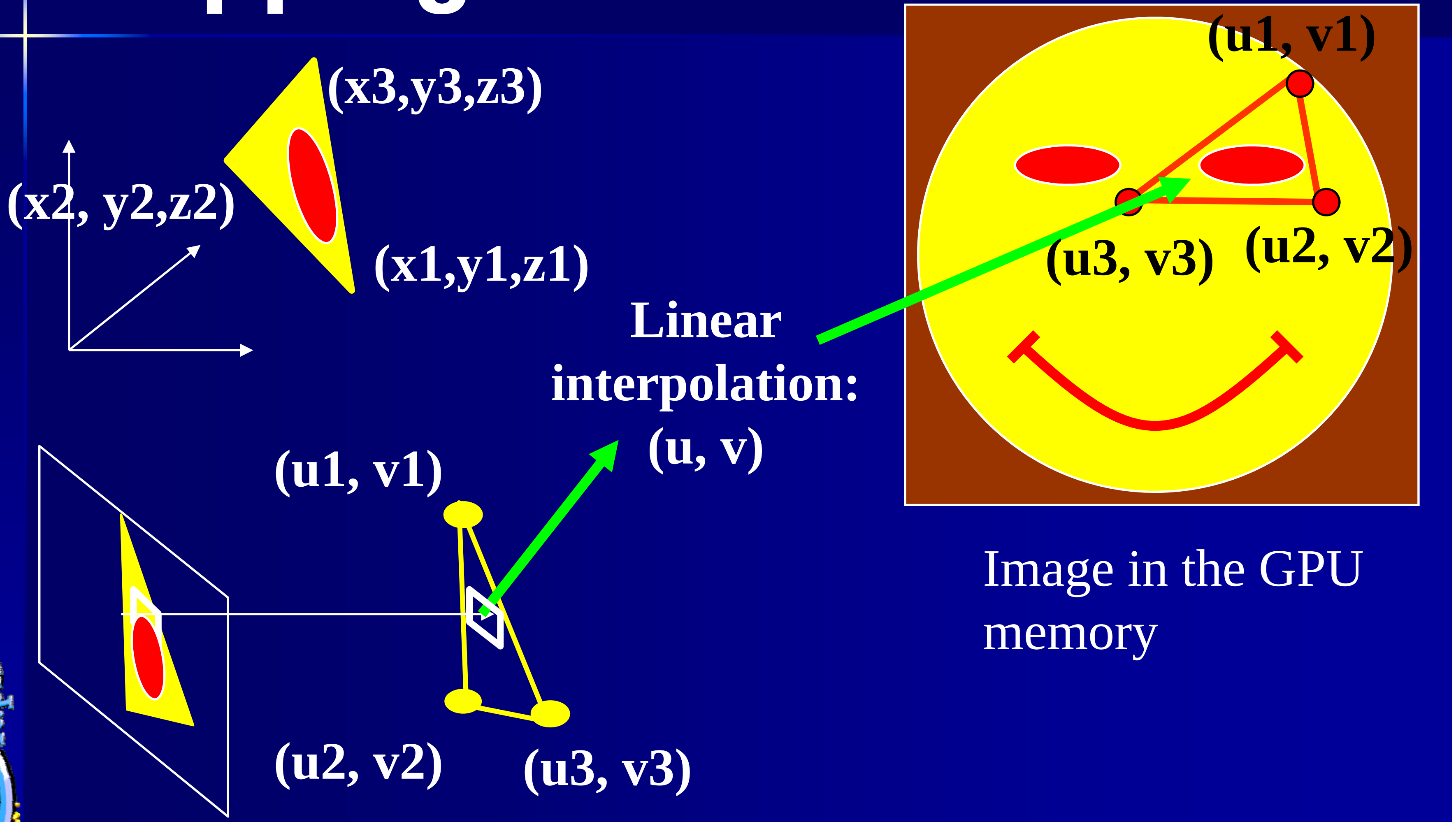
Screen transformation + Rasterization

display

Texture mapping



Hw support for texture mapping



(x3,y3,z3)

(x2, y2,z2)

(x1,y1,z1)

Linear interpolation: (u, v)

(u1, v1)

(u2, v2)

(u3, v3)

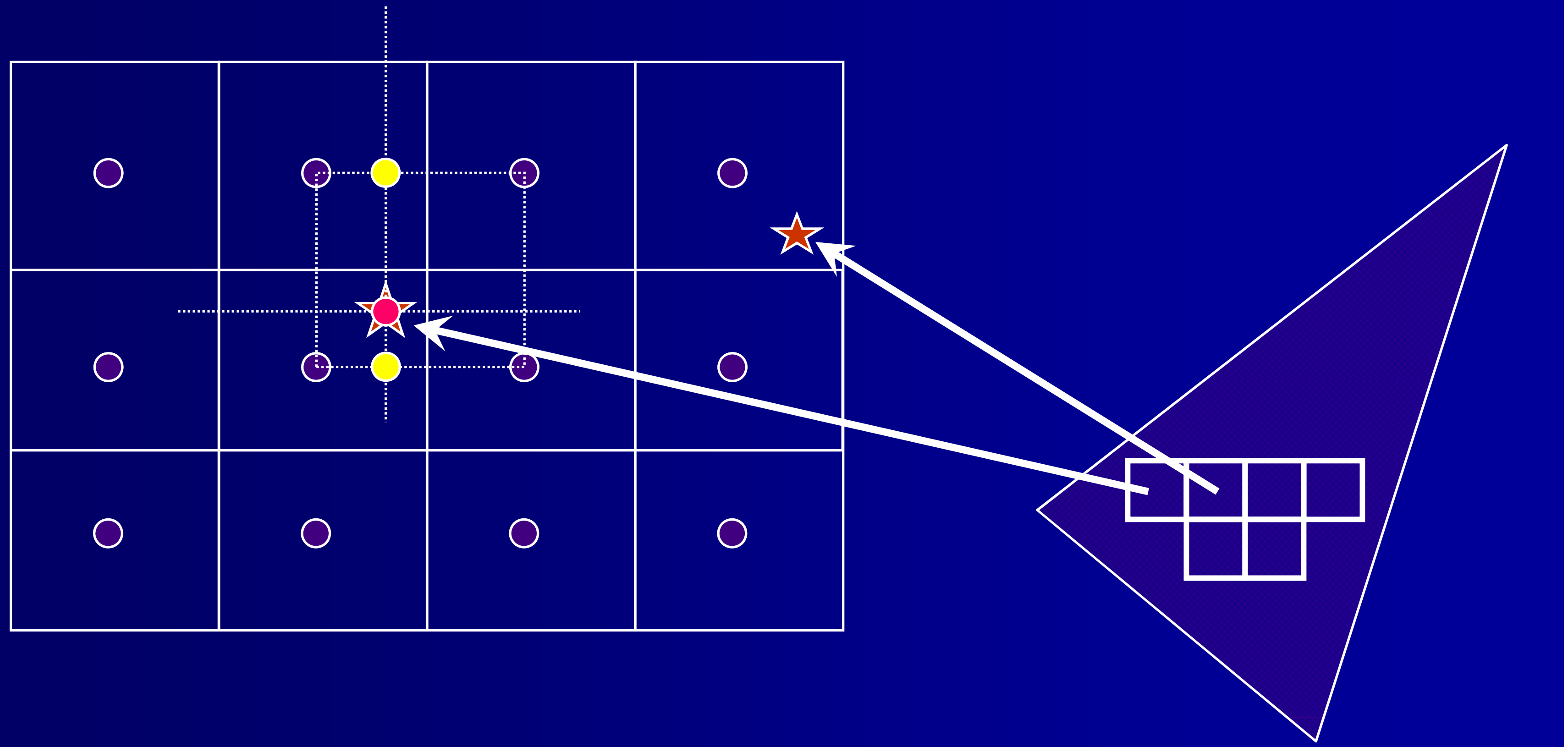
(u1, v1)

(u3, v3)

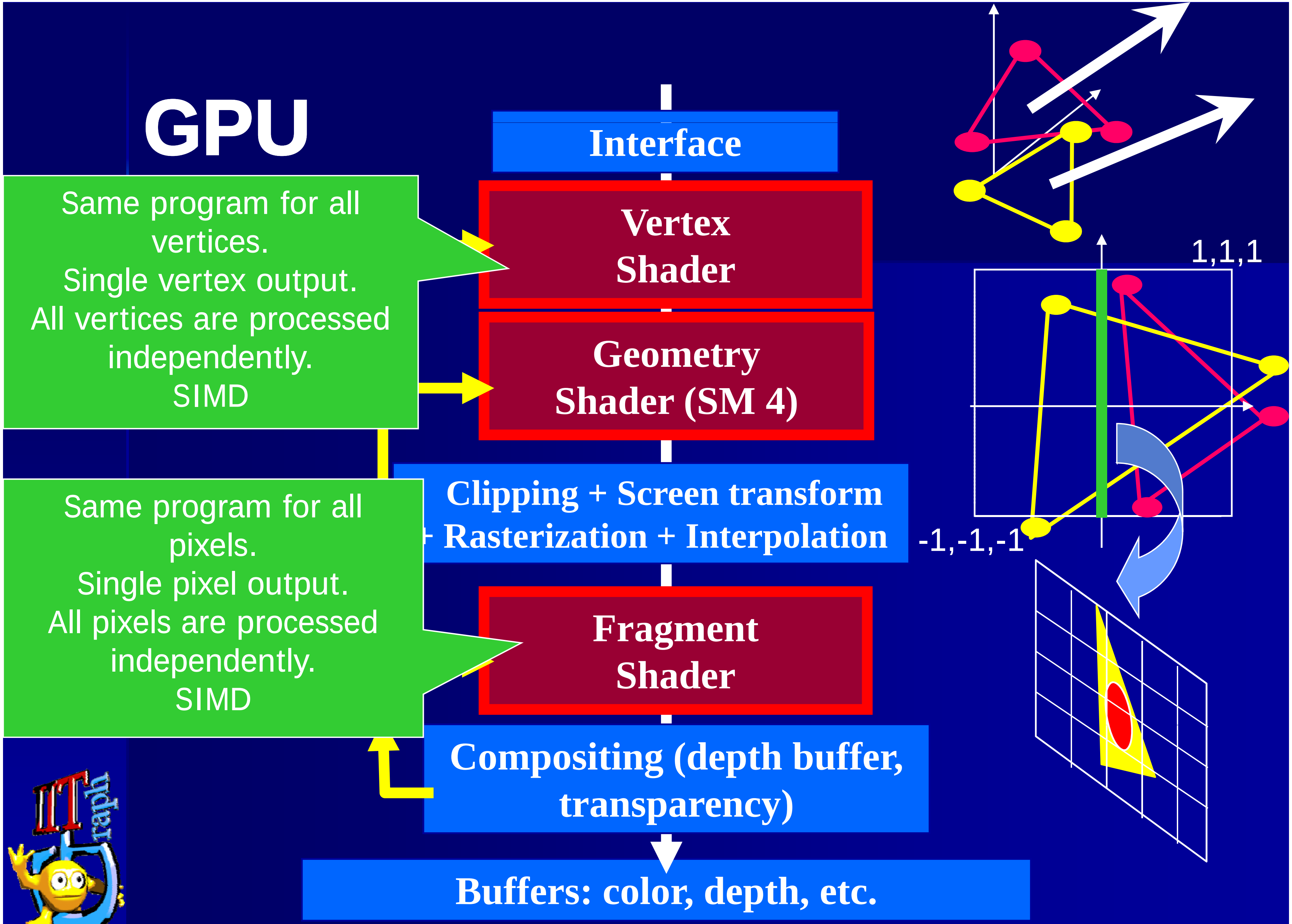
(u2, v2)

Image in the GPU memory

Texture filtering



GPU



Interface

Vertex Shader

Geometry Shader (SM 4)

Clipping + Screen transform + Rasterization + Interpolation

Fragment Shader

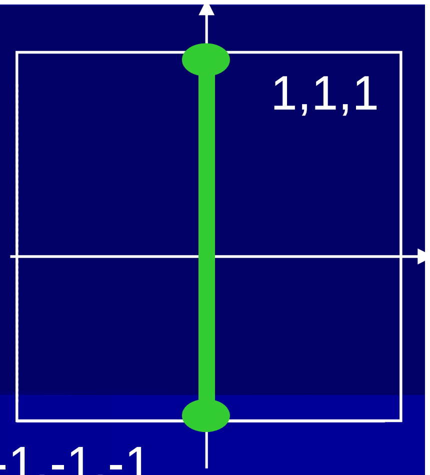
Compositing (depth buffer, transparency)

Buffers: color, depth, etc.

Same program for all vertices.
Single vertex output.
All vertices are processed independently.
SIMD

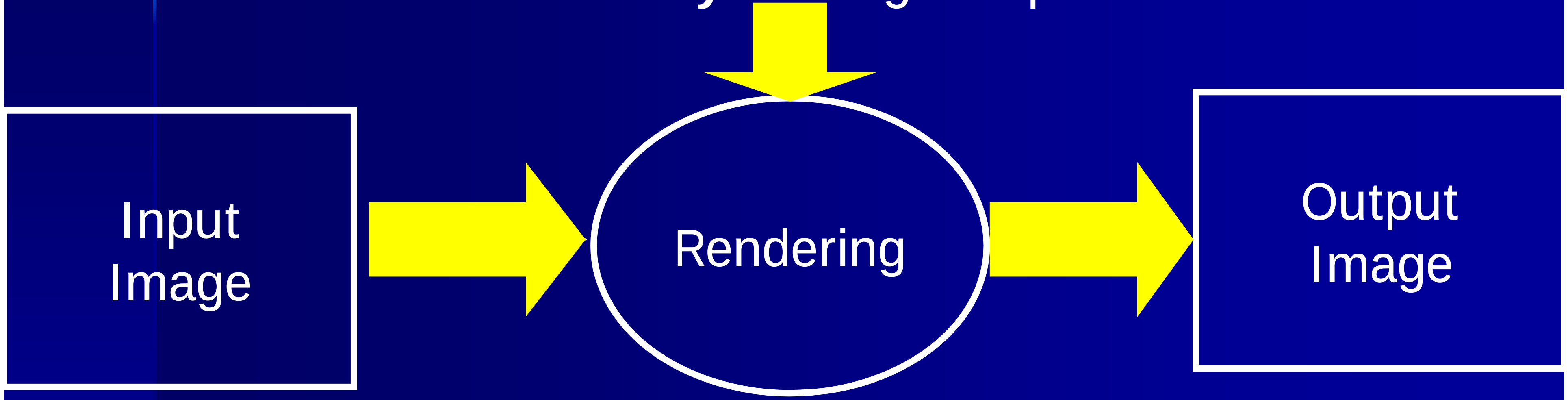
Same program for all pixels.
Single pixel output.
All pixels are processed independently.
SIMD

Image processing



Geometry: triangles

Geometry: "full screen" quad



Input Image

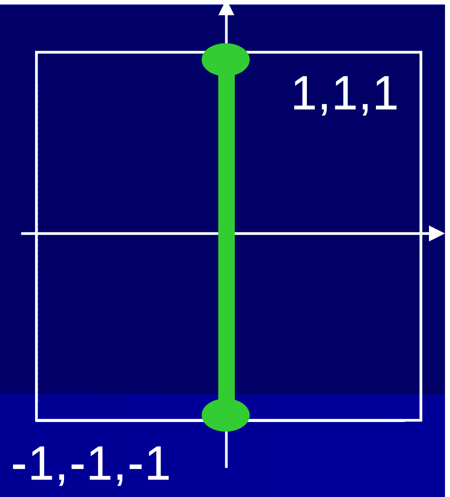
Rendering

Output Image

Texture

Texture or Raster Memory

Image processing



Input
Image

Texture

Full screen quad (CPU):

```
glViewport(0, 0, HRES, VRES)
glBegin(GL_QUADS);
  glVertex4f(-1,-1, 0, 1);
  glVertex4f(-1, 1, 0, 1);
  glVertex4f( 1, 1, 0, 1);
  glVertex4f( 1,-1, 0, 1);
glEnd( );
```

Vertex shader (Cg):

```
void VS(in float4 inPos  : POSITION,
             out float4 hPos : POSITION) {
    hPos = inPos;
}
```

Fragment shader (Cg):

```
void FS( in float2 index : WPOS,
            uniform samplerRECT In
            out float4 outColor : COL
    outColor = F(index);
}
```

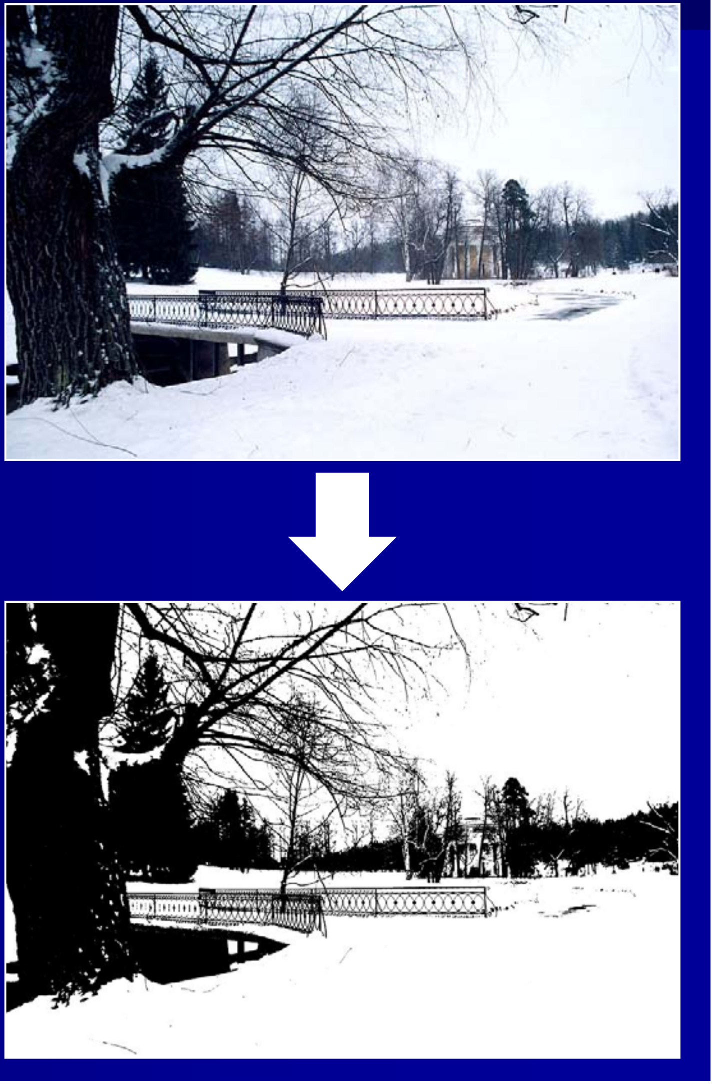
Output
Image

Texture or
Raster Memory

How to compute a single output pixel from the input pixels. Gathering!

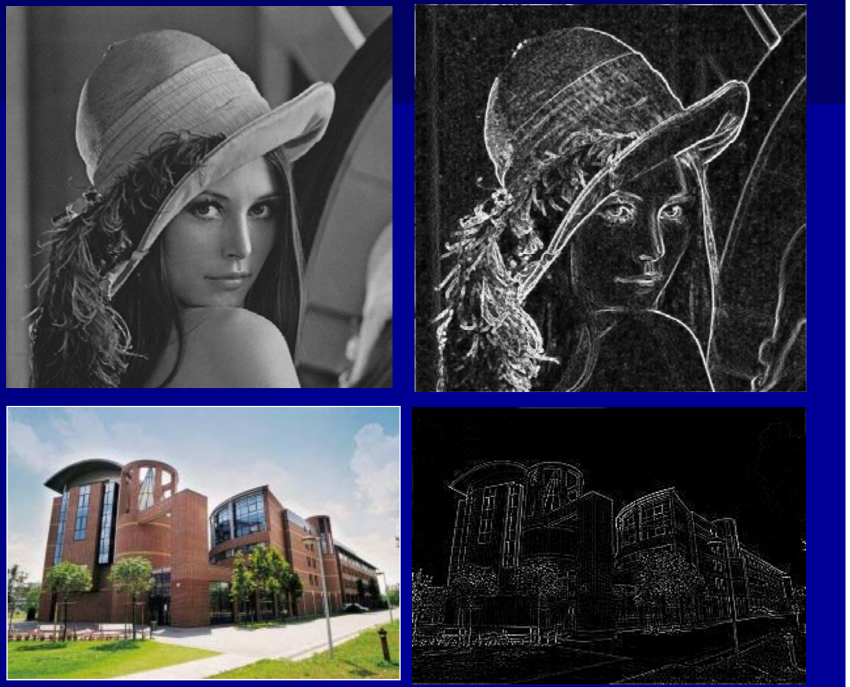
Luminance transformation and thresholding

$$I = \begin{bmatrix} r & g & b \end{bmatrix} \begin{bmatrix} 0.21 \\ 0.39 \\ 0.4 \end{bmatrix}$$



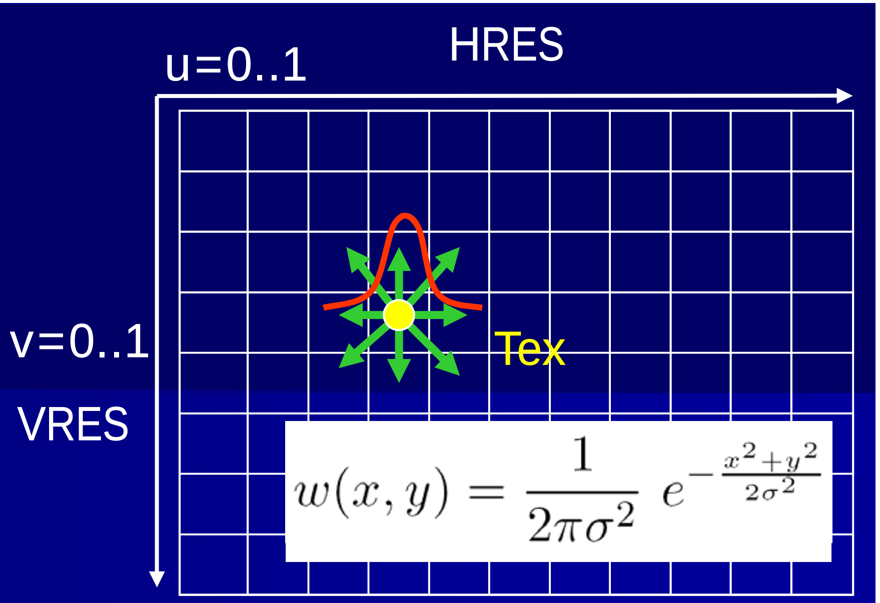
```
void FS(
     in float2 index : WPOS,
     uniform samplerRECT Image,
     uniform float threshold,
     out float4 outColor : COLOR )
{
  float3 color = texRECT(Image, index);
  float I = dot(color, float3(0.21, 0.39, 0.4));
  outColor = I > threshold ?
              float4(1.0) : float4(0.0);
}
```

Edge detection



```
void FS(
     in float2 index : WPOS,
     uniform samplerRECT Image,
     out float4 outColor : COLOR )
{
    float2 dx = float2(1, 0);
    float2 dy = float2(0, 1);
    float dIdx = (texRECT(Image, index+dx)–textRECT(Image, index–dx))/2;
    float dIdy = (texRECT(Image, index+dy)–textRECT(Image, index–dy))/2;
    float gradabs = sqrt(dIdx * dIdx + dIdy * dIdy);
    outColor = float4(gradabs, gradabs, gradabs, 1);
}
```

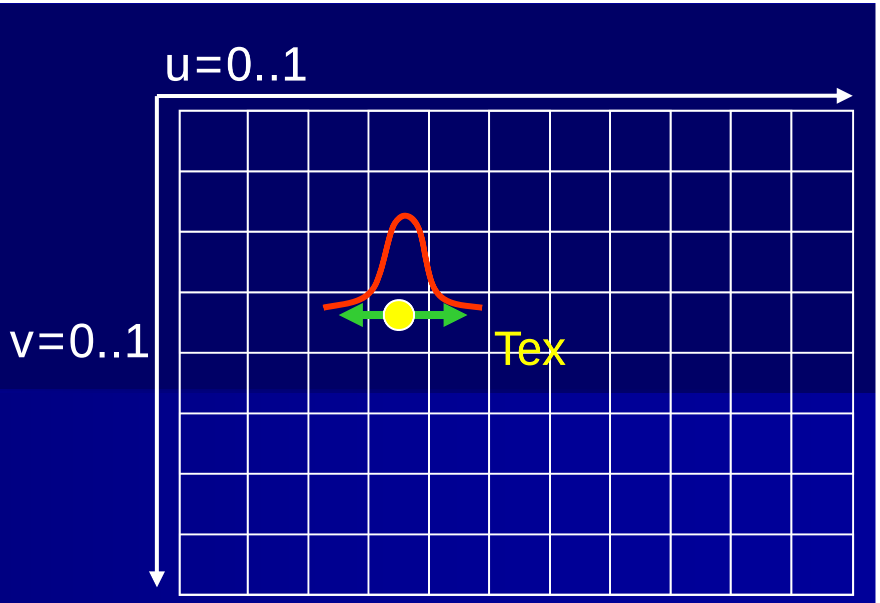
Filtering



$$w(x,y) = \frac{1}{2\pi\sigma^2}\, e^{-\frac{x^2+y^2}{2\sigma^2}}$$

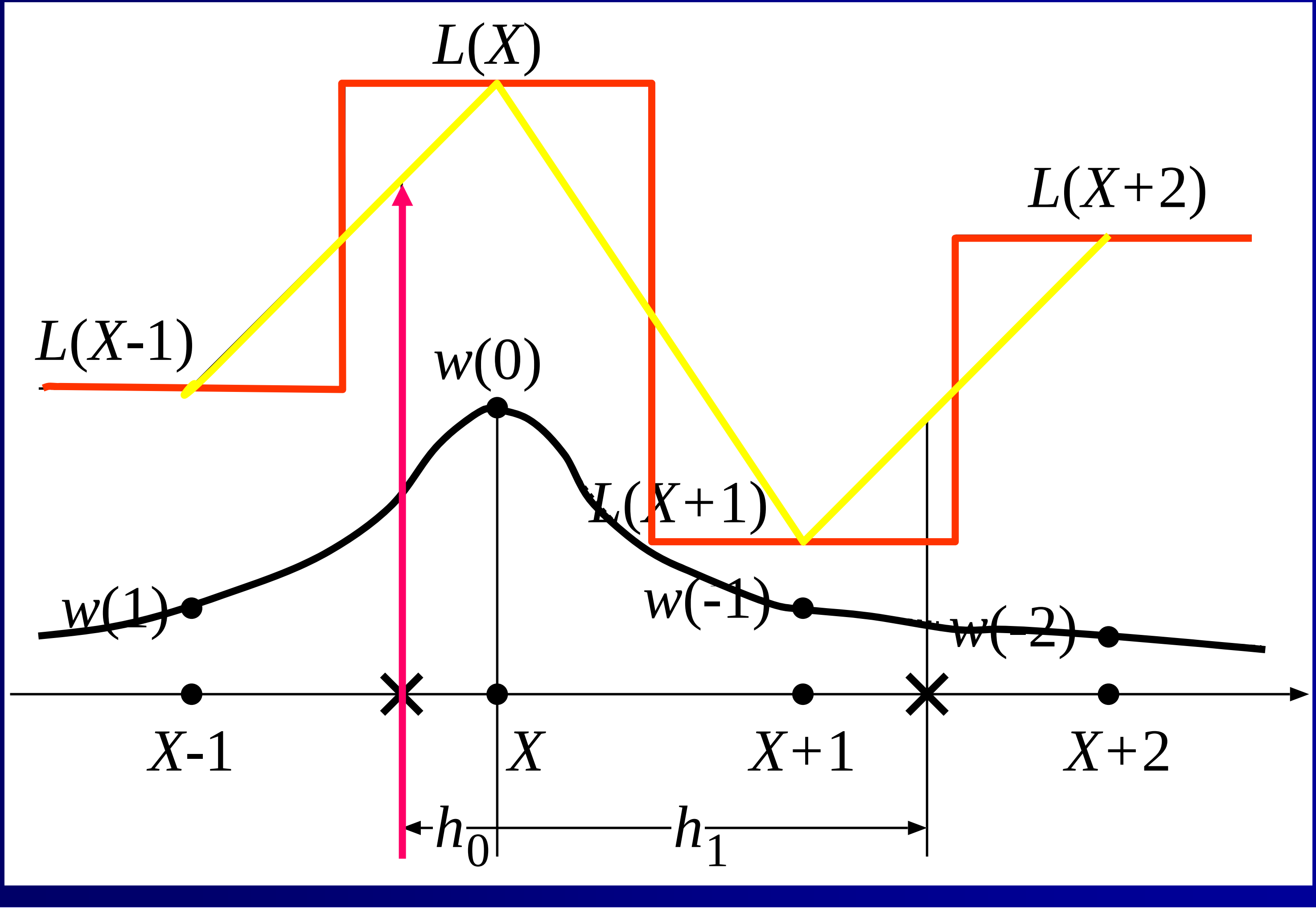
```
void FS(
      in float2 index : WPOS,
      uniform samplerRECT Image,
      uniform int N,                          // kernel width
      uniform float sigma2,
      out float3 outColor : COLOR )
{
    outColor = float4(0, 0, 0, 0);
    for(int i = -N/2, i < N/2; i++) for(int j = -N/2, j < N/2; j++) {
          float2 duv =float2(i, j);
          float w = exp( -dot(duv, duv)/2/sigma2 ) / 6.28 / sigma2;
          outColor += texRECT(Image, index- duv) * w;
    }
}
```

Separation of coordinates

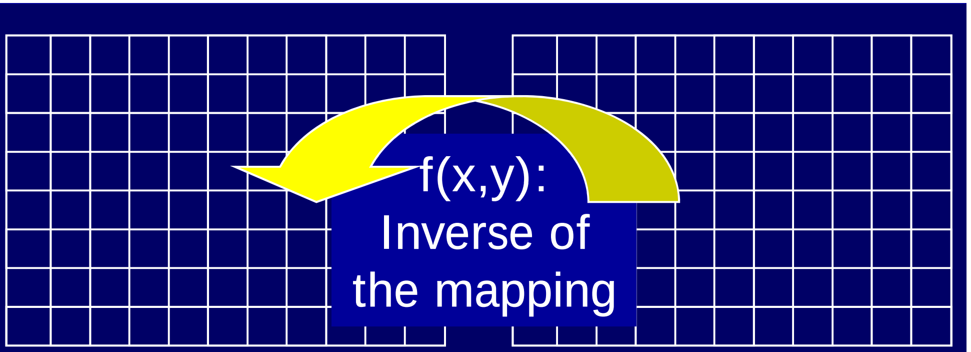


```
void HFS(
     in float2 index : WPOS,
     uniform samplerRECT Image,
     uniform int N,                          // kernel width
     uniform float sigma2,
     out float3 outColor : COLOR )
{
   outColor = float4(0, 0, 0, 0);
   for(int i = -N/2, i < N/2; i++) {
        float w = exp( -i * i/2/sigma2 ) / sqrt(6.28 * sigma2);
        outColor += texRECT(Image, index - float2(i, 0)) * w;
   }
}
```

Exploitation of bi-linear filtering



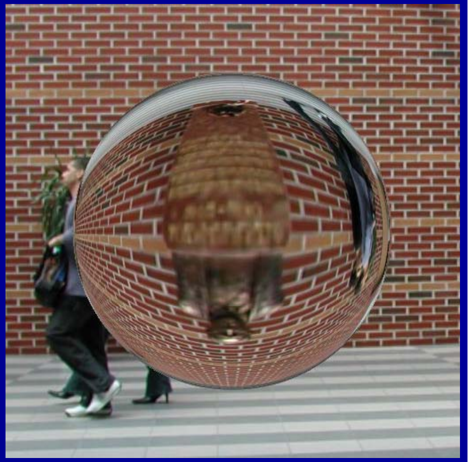
Distortions



f(x,y): Inverse of the mapping

Source

Target

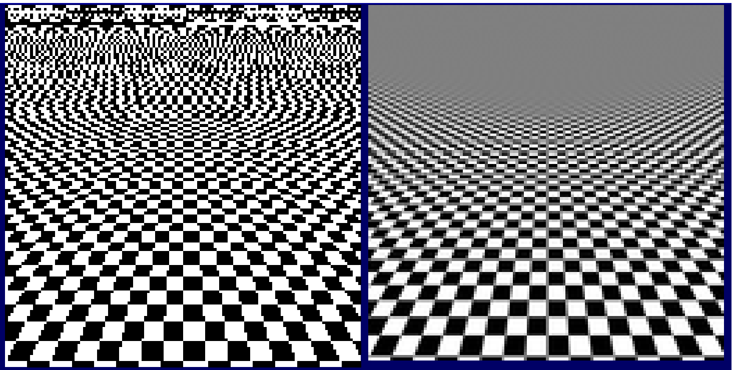


Texture mapping is a homogeneous linear distortion filter!

```
float2 f( float2 outPixelCoord )
{
   float2 inPixelCoord = ...
   return inPixelCoord;
}
```

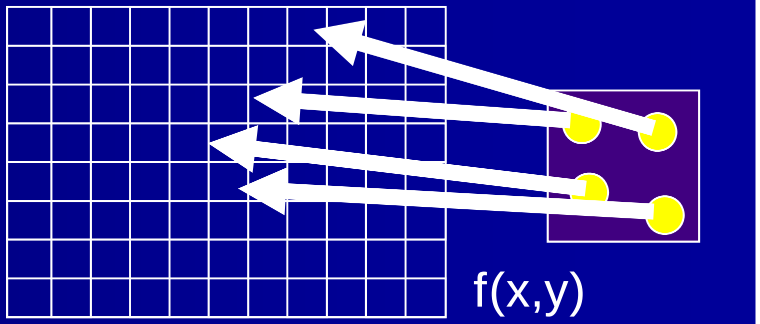
```
void FS(
     in float2 index : WPOS,
     uniform samplerRECT Image,
     out float3 outColor : COLOR )
{
   outColor = texRECT(Image, f(index) ).rgb;
}
```

Distortions with anti-aliasing



Uniform supersamples:

- Regular grid
- Poisson disk
- Low-discrepancy
- Random



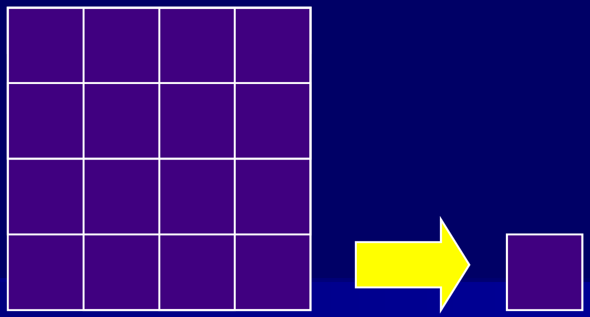
f(x,y)

Source

Target pixel

```
void FS(
      in float2 index : WPOS,
      uniform samplerRECT Image,
      uniform float2 offsets[4],           // in [0,1]^2
      out float3 outColor : COLOR )
{
    outColor    = texRECT(Image, f(index+ offsets[0])).rgb;
    outColor += texRECT (Image, f(index+ offsets[1])).rgb;
    outColor += texRECT (Image, f(index+ offsets[2])).rgb;
    outColor += texRECT (Image, f(index+ offsets[3])).rgb;
    outColor /= 4;
}
```

Averaging (Reduction)

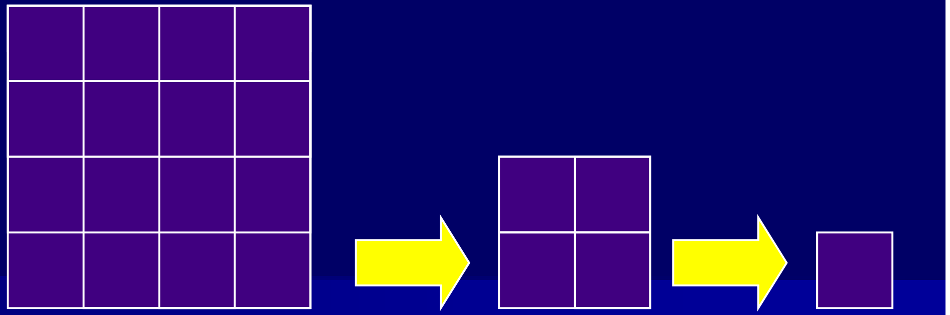


CPU:

```
glViewport(0, 0, 1, 1);
```

```
void FS(
    uniform samplerRECT Image,
    uniform int2 ImageRes,
    out float3 outColor : COLOR )
{
    outColor = 0;
    for(int x=0; x<ImageRes.x; ++x)
        for(int y=0; y<ImageRes.y; ++y) {
            outColor += texRECT (Image, float2(x, y));
    outColor /= ImageRes.x * ImageRes.y;
}
```

Averaging (Reduction)



CPU:

```
for(RES = image resolution/2; RES > 1; RES /= 2) {
    glViewport(0, 0, RES, RES);
    Draw full screen quad;
    Texture ping-pong;
}
```

```
void FS(
     in float2 index : WPOS,
     uniform samplerRECT Image,
     out float3 outColor : COLOR )
{
   outColor   = texRECT(Image, 2*index).rgb;
   outColor += texRECT(Image, 2*index + float2(1, 0)).rgb;
   outColor += texRECT(Image, 2*index + float2(1, 1)).rgb;
   outColor += texRECT(Image, 2*index + float2(0, 1)).rgb;
   outColor /= 4;
}
```

Exploitation of the built-in bi-linear filter

CPU:

```
for(RES = image resolution/2; RES > 1; RES /= 2) {
    glViewport(0, 0, RES, RES);
    Draw full screen quad;
    Texture ping-pong;
}
```

Fragment shader:

```
void FS(
    in float2 index : WPOS,
    uniform samplerRECT Image,
    out float3 outColor : COLOR )
{
   outColor = texRECT(Image, 2*index + float2(0.5, 0.5));
}
```

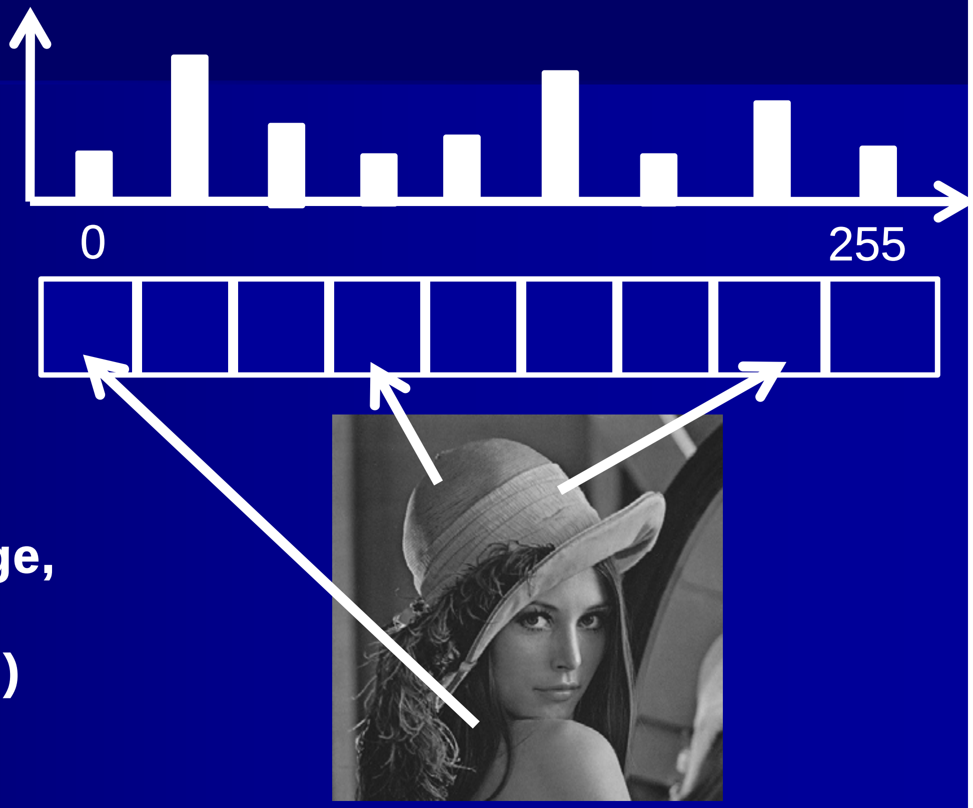
Histogram

CPU:

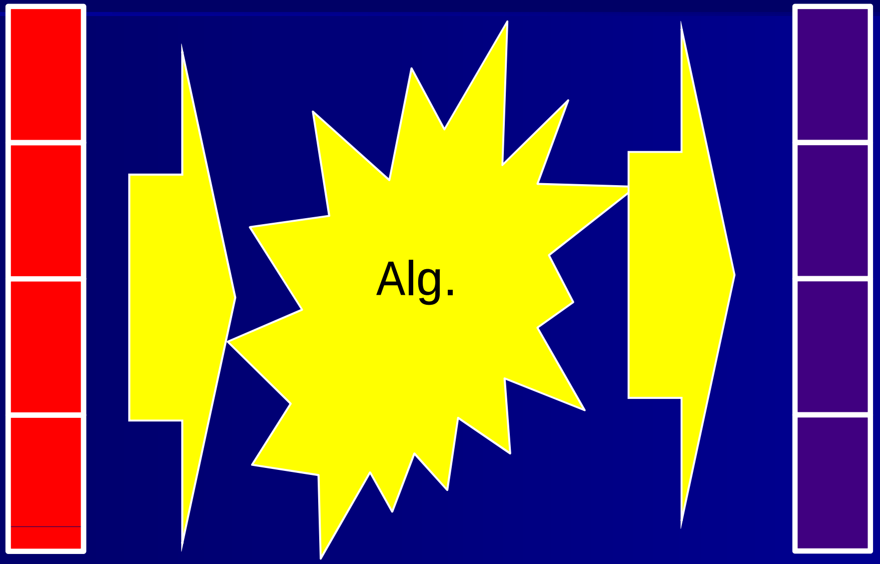
```
glViewport(0, 0, 256, 1);
Draw full screen quad;
```

Fragment shader:

```
void FS(
     in float2 index : WPOS,
     uniform samplerRECT Image,
     uniform int2 ImageRes,
     out float outColor : COLOR )
{
   outColor = 0;
   for(int x=0; x<ImageRes.x; ++x) for(int y=0; y<ImageRes.y; ++y) {
      float col = texRECT (Image, float2(x, y));
      if (index.x <= col && col < index.x + 1) outColor++;
   }
}
```



Gather versus Scatter



inputs

outputs

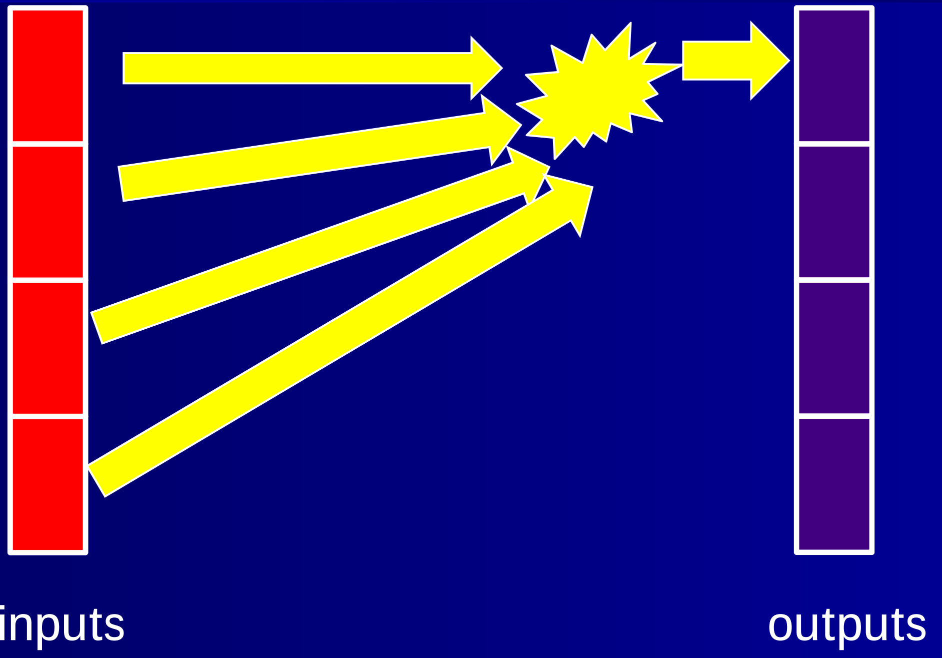
Gather:

```
for each output
    for each relevant input
        Add input's contrib. to output
```

Scatter:

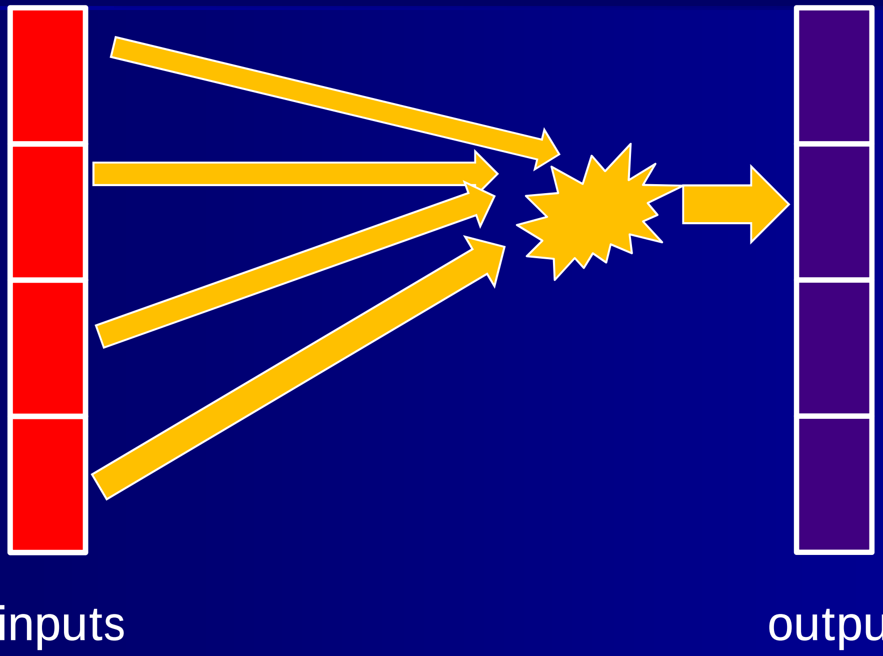
```
for each input
    for each relevant output
        Add input's contrib. to output
```

Gather



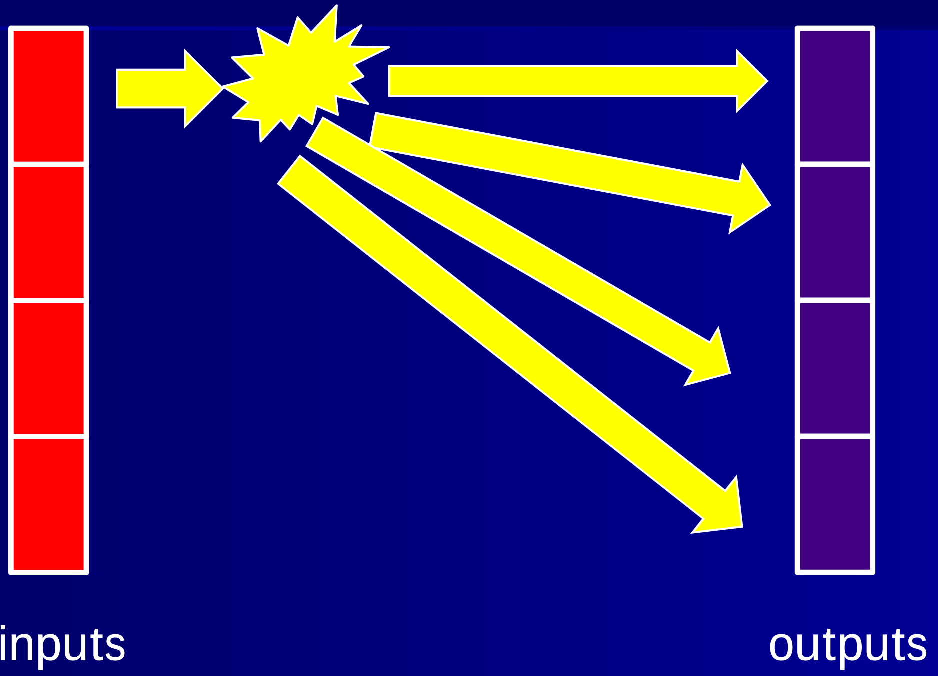
```
for each output
    for each relevant input
        Add input's contrib. to output
```

Gather



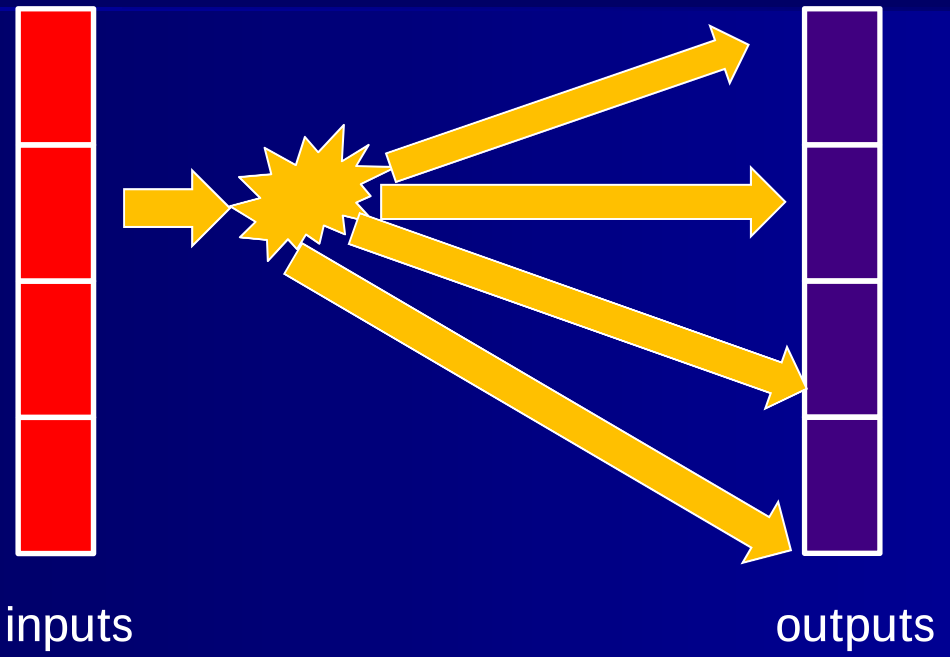
```
for each output
    for each relevant input
        Add input's contrib. to output
```

Scatter:
Not on Fragment Shader



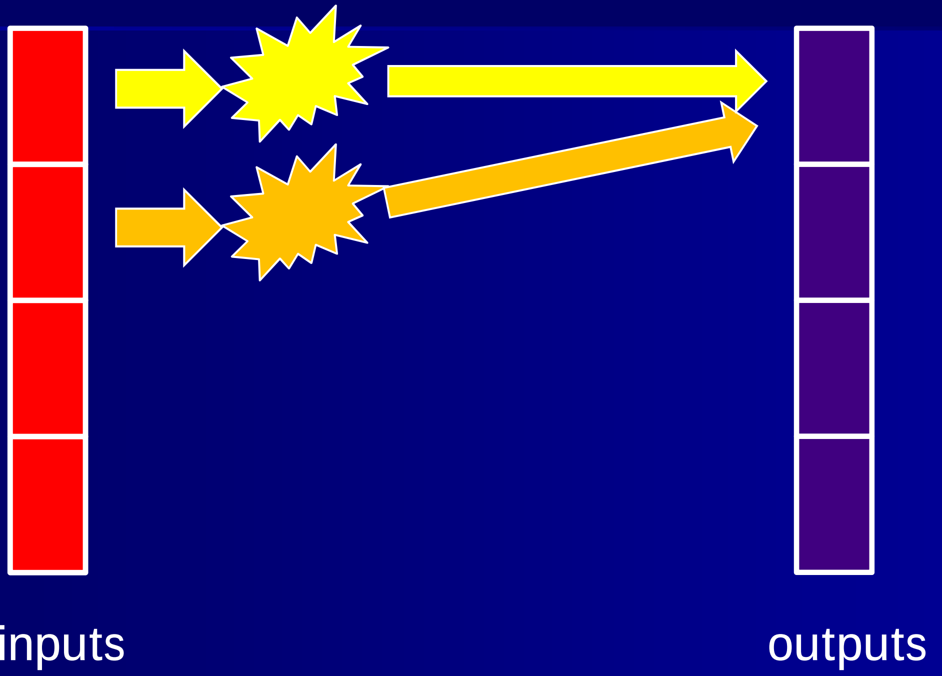
```
for each input
  for each relevant output
    Add input's contrib. to output
```

Scatter:
Not on Fragment Shader



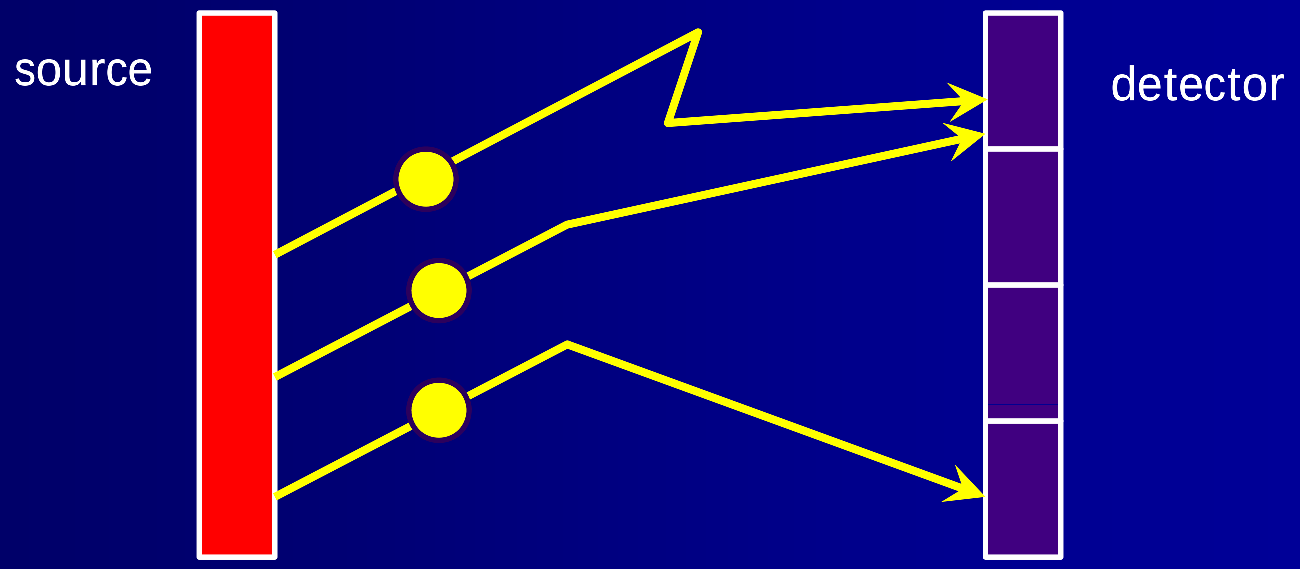
```
for each input
    for each relevant output
        Add input’s contrib. to output
```

Scatter: Not on Fragment Shader



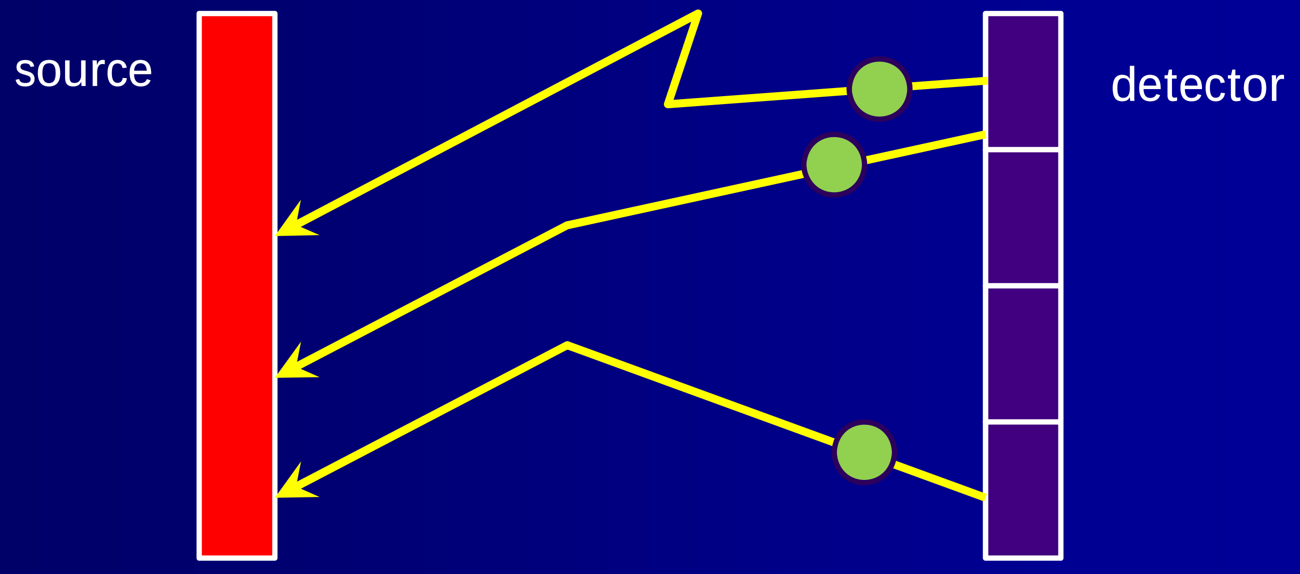
Write collisions: atomic operations or synchronization

Can you prefer gather? Particle transport



e.g. photons

Can you prefer gather? Particle transport

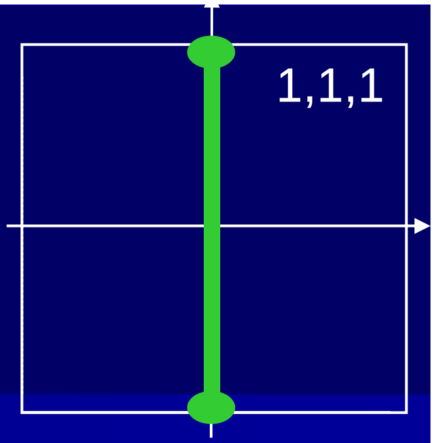


importons

Histogram

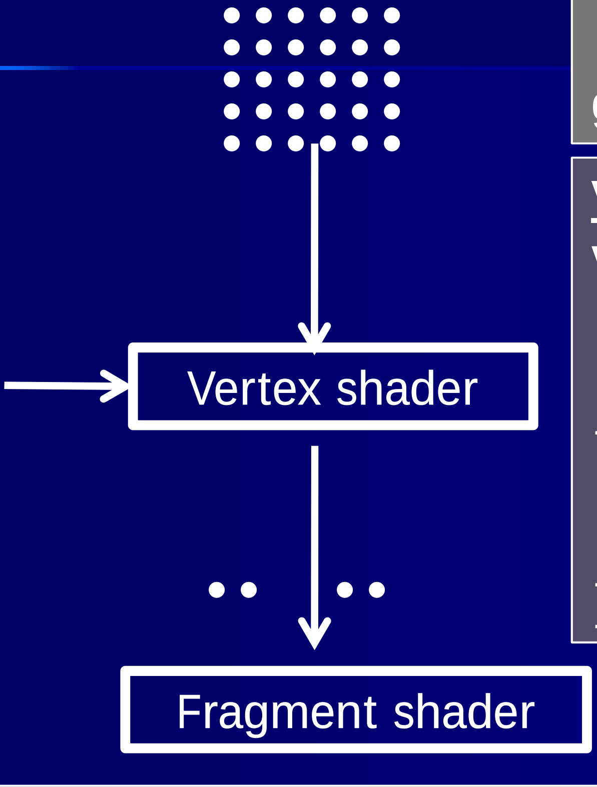
CPU:

```
glViewport(0, 0, 256, 1);
glBegin(GL_POINTS);
for(x=0; x < RX; x++)
    for(y=0; y < RY; y++)
        glVertex2f(x/RX, y/RY);
glEnd( );
```



Vertex shader

```
void VS( in float4 position : POSITION,
         uniform samplerRECT Image,
         out float4 hPos : POSITION )
{
    float col = texRECT(Image, position.xy);
    hPos = float4(2*(col - 0.5), 0, 0, 1);
}
```



Additive blending

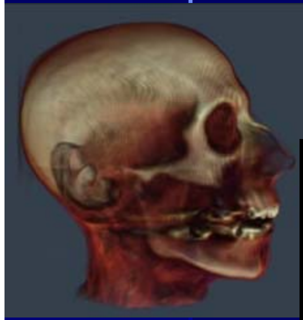
1	2	15	6	4	9	31

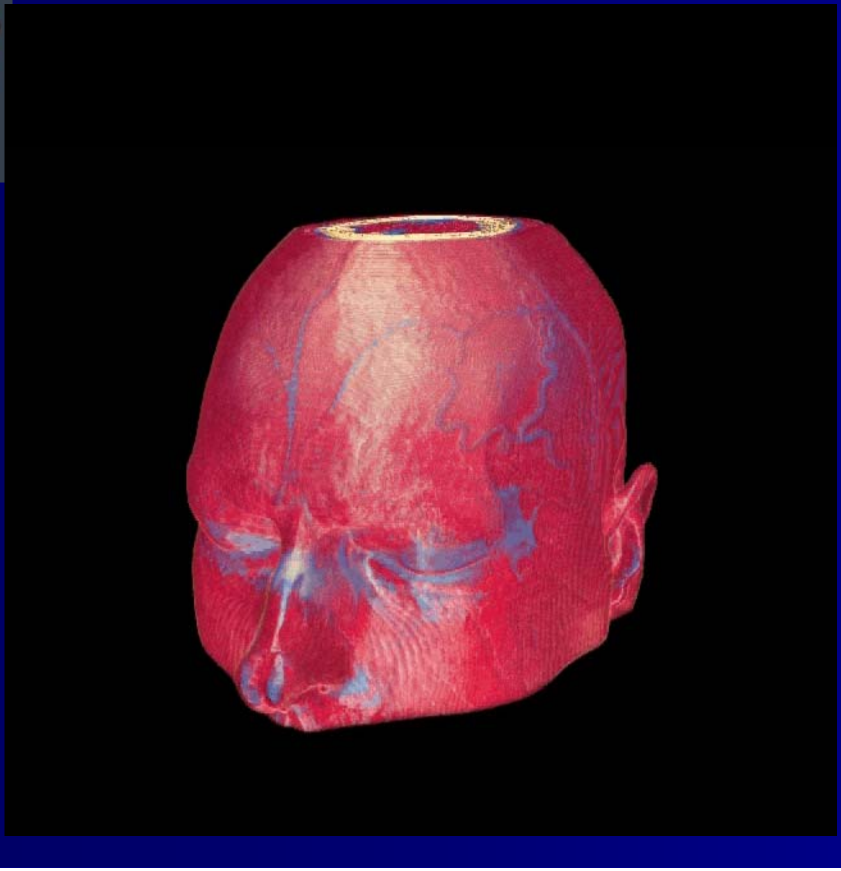
Fragment shader

```
void FS( out float4 outColor : COLOR )
{
    outColor = float4(1, 1, 1, 1);
}
```

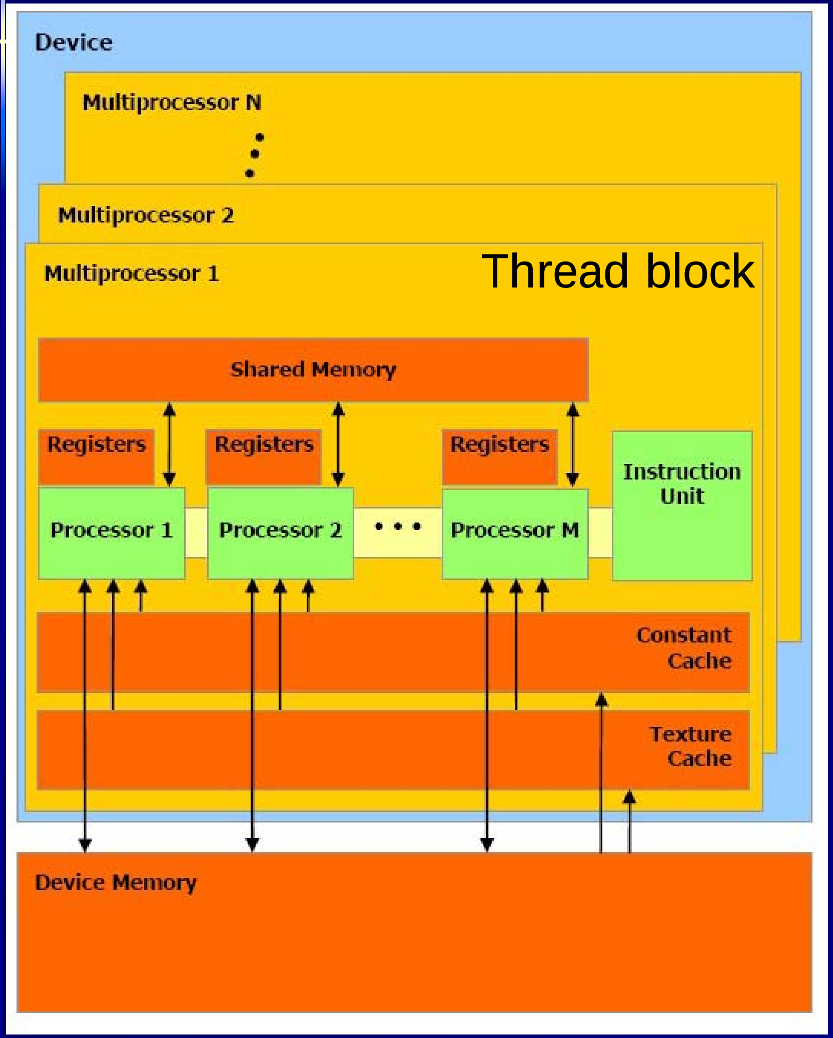
Shader programming

Shader programming



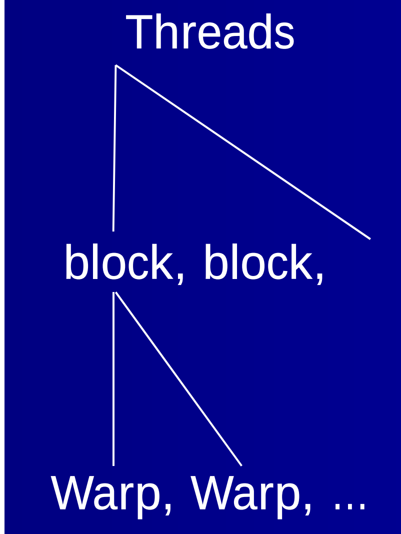


CUDA (OpenCL)



GPU

Kernel program:

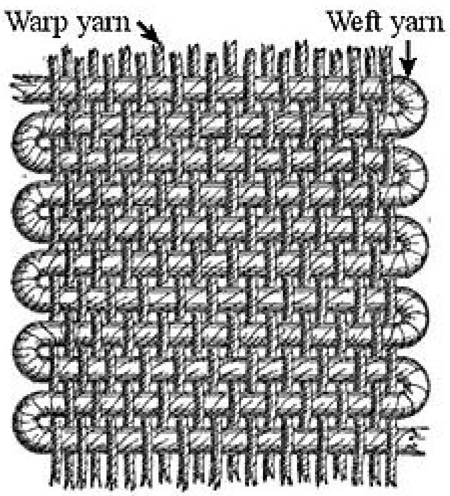


Threads

block, block,

Warp, Warp, ...

SIMD



Shared memory

SIMD execution

Add two N element vectors

Runs on the GPU, but can be called from the CPU

```
__global__ void AddVectorGPU( float *C, float *A, float *B, int N ) {
   int i = blockIdx.x * blockDim.x + threadIdx.x; // szálazonosító
   if (i < N)
        C[i] = A[i] + B[i];
}
```

0 ,..., gridDim.x-1

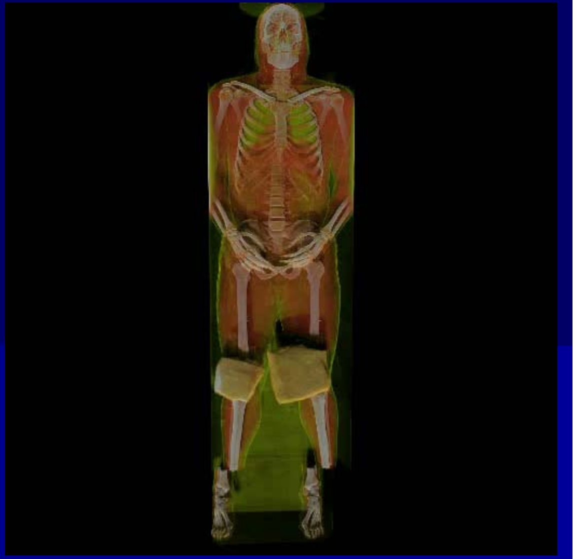
0 ,..., blockDim.x-1

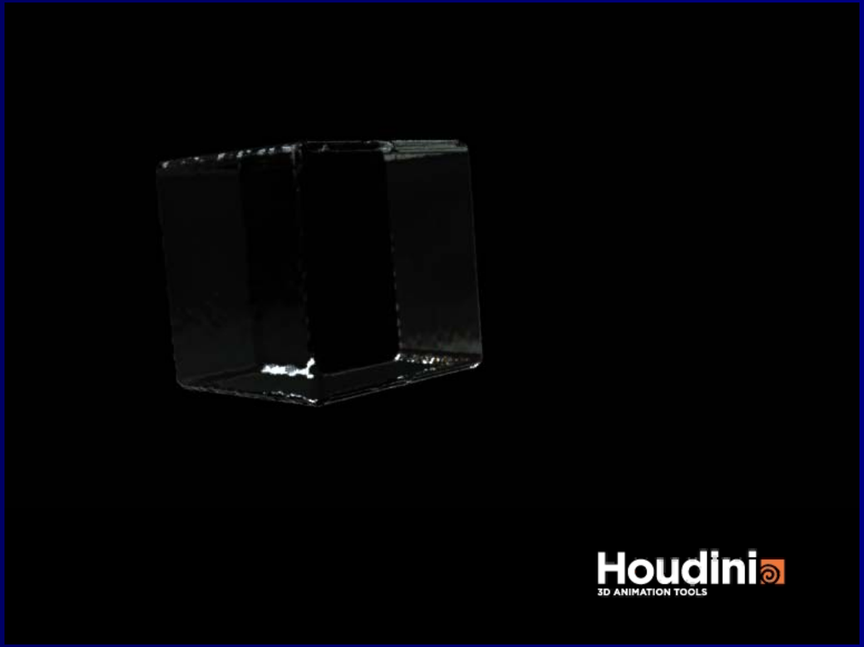
```
float C[100000], A[100000], B[100000];

int main ( ) {
    ...
    int N = 100000;
    ...
    int blockDim = 256;     // #threads in a block: 128, 256, 512
    int gridDim = (N + blockDim – 1) / blockDim;       // #blocks
    AddVectorGPU<<<gridDim, blockDim>>>(C, A, B, N);
    ...
}
```

GPGPU

$$\vec{\omega}\cdot\nabla L = -\sigma_t L\left(\vec{x},\vec{\omega}\right) + \sigma_t \int_{\Omega} L\left(\vec{x},\vec{\omega}'\right) P(\vec{\omega},\vec{\omega}')d\omega'$$



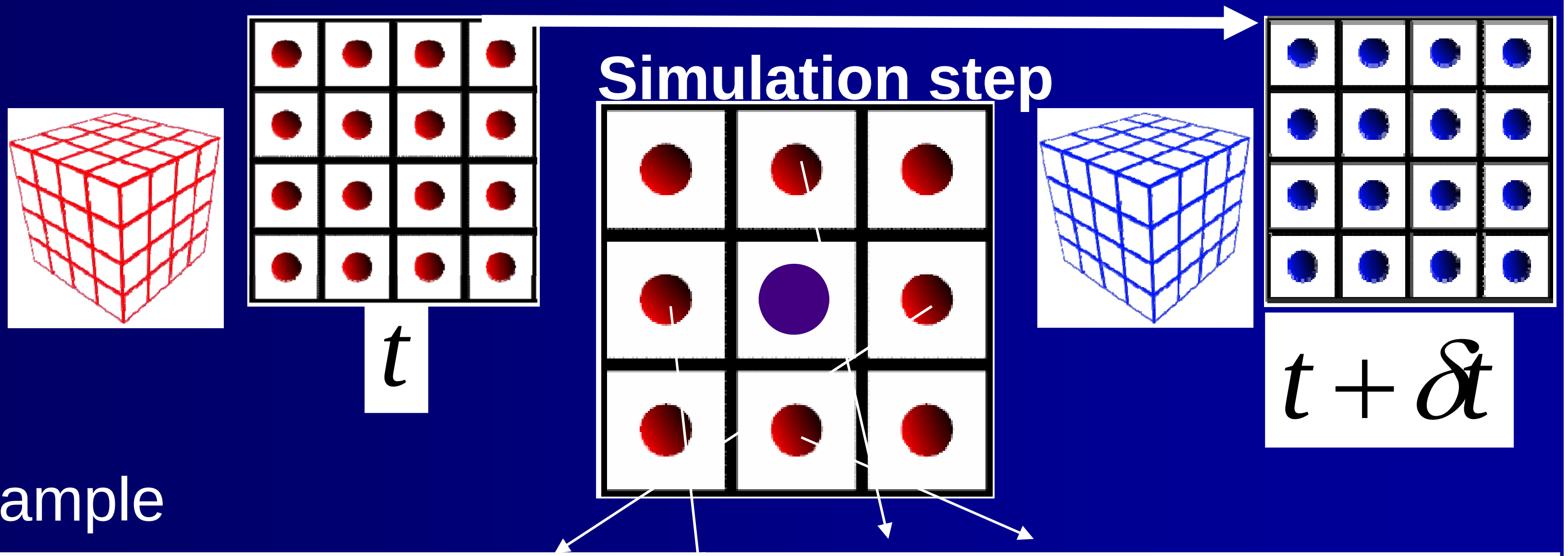


$$\frac{d\vec{u}}{dt} = -\left(\vec{u}\cdot\nabla\right)\vec{u} - \frac{1}{\rho}\nabla p + v\nabla^2\vec{u} + \vec{F}$$

$$\nabla\cdot\vec{u} = 0$$

Numerical integration

$$\vec{u}(t+\delta t)=\vec{u}(t)-\left(\vec{u}\cdot\nabla\right)\vec{u}\,\delta t-\frac{1}{\rho}\nabla p\,\delta t+v\nabla^{2}\vec{u}\,\delta t+\vec{F}\,\delta t$$



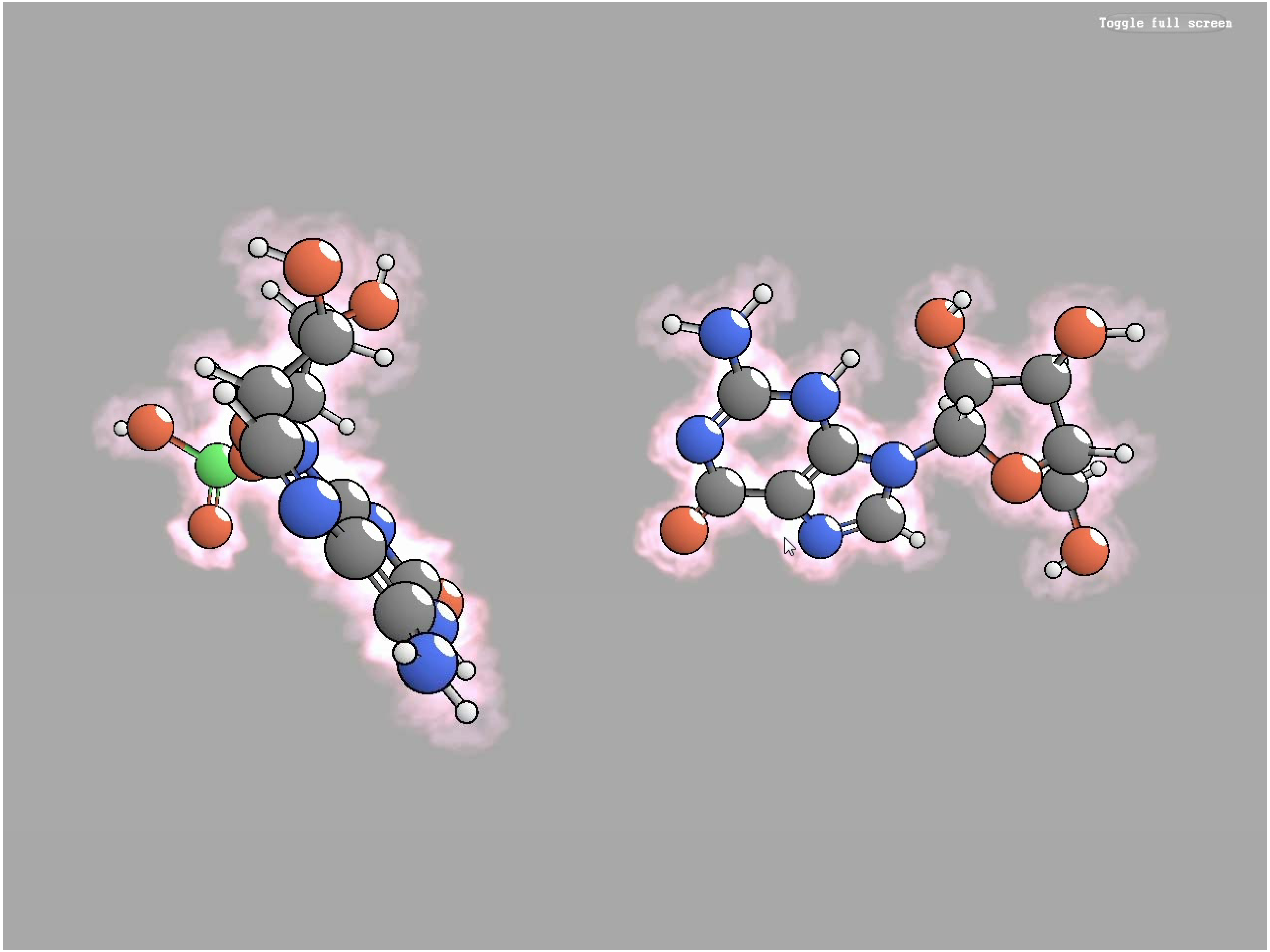
Example

$$\nabla\cdot\vec{u}=\mathrm{div}\,\vec{u}=\frac{\partial\vec{u}_x}{\partial x}+\frac{\partial\vec{u}_y}{\partial y}+\frac{\partial\vec{u}_z}{\partial z}=\frac{\vec{u}_x^{\,\mathrm{i+1,j,k}}-\vec{u}_x^{\,\mathrm{i-1,j,k}}}{2\delta x}+\frac{\vec{u}_y^{\,\mathrm{i,j+1,k}}-\vec{u}_y^{\,\mathrm{i,j-1,k}}}{2\delta y}+\frac{\vec{u}_z^{\,\mathrm{i,j,k+1}}-\vec{u}_z^{\,\mathrm{i,j,k\text{-}1}}}{2\delta z}$$

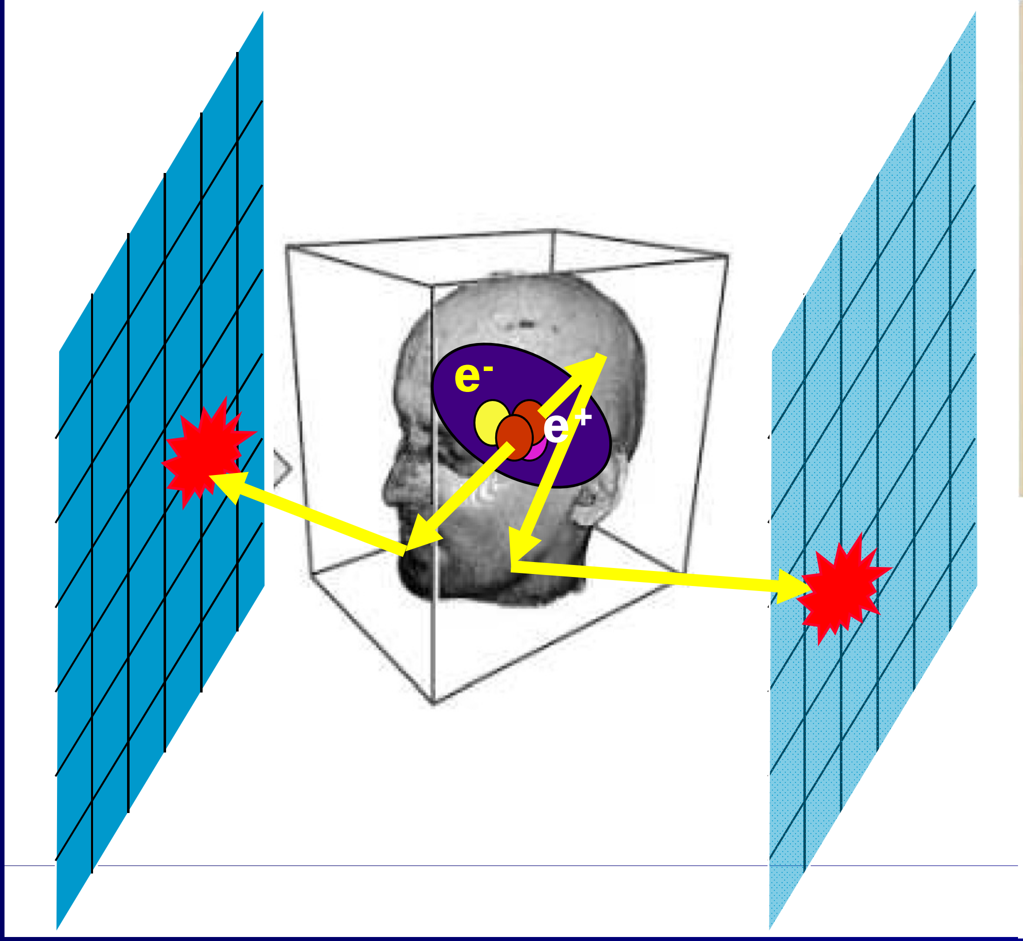
N-body simulation

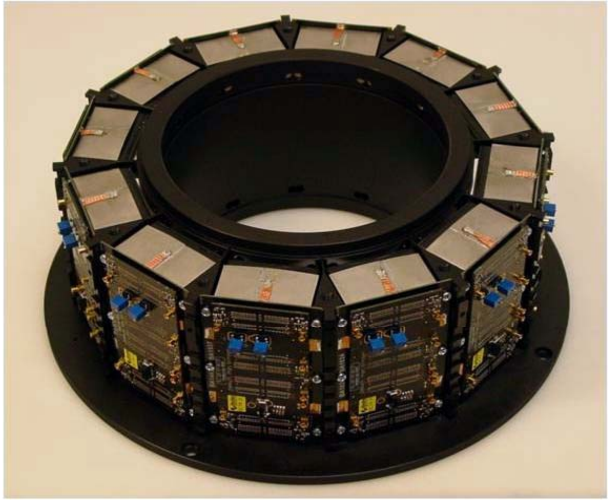
- Position ***p*** + velocity ***v*** → forces ***f*** (gravity, Columb, van der Waals, Pauli)
- Forces → acceleration *a*
- Acceleration → updated position+velocity

```
f = float3(0, 0, 0);
for(int i = 0; i < N; i++)
    if (i != index) f += Force(p[i], p[index]);
float3 a = f/m;
v[index] += a * dt;
p[index] += v[index] * dt;
```


Toggle full screen

Positron Emission Tomography





Mediso NanoPET^{TM}/CT

Mediso PET / CT