CUDA FOR GRAPHICS

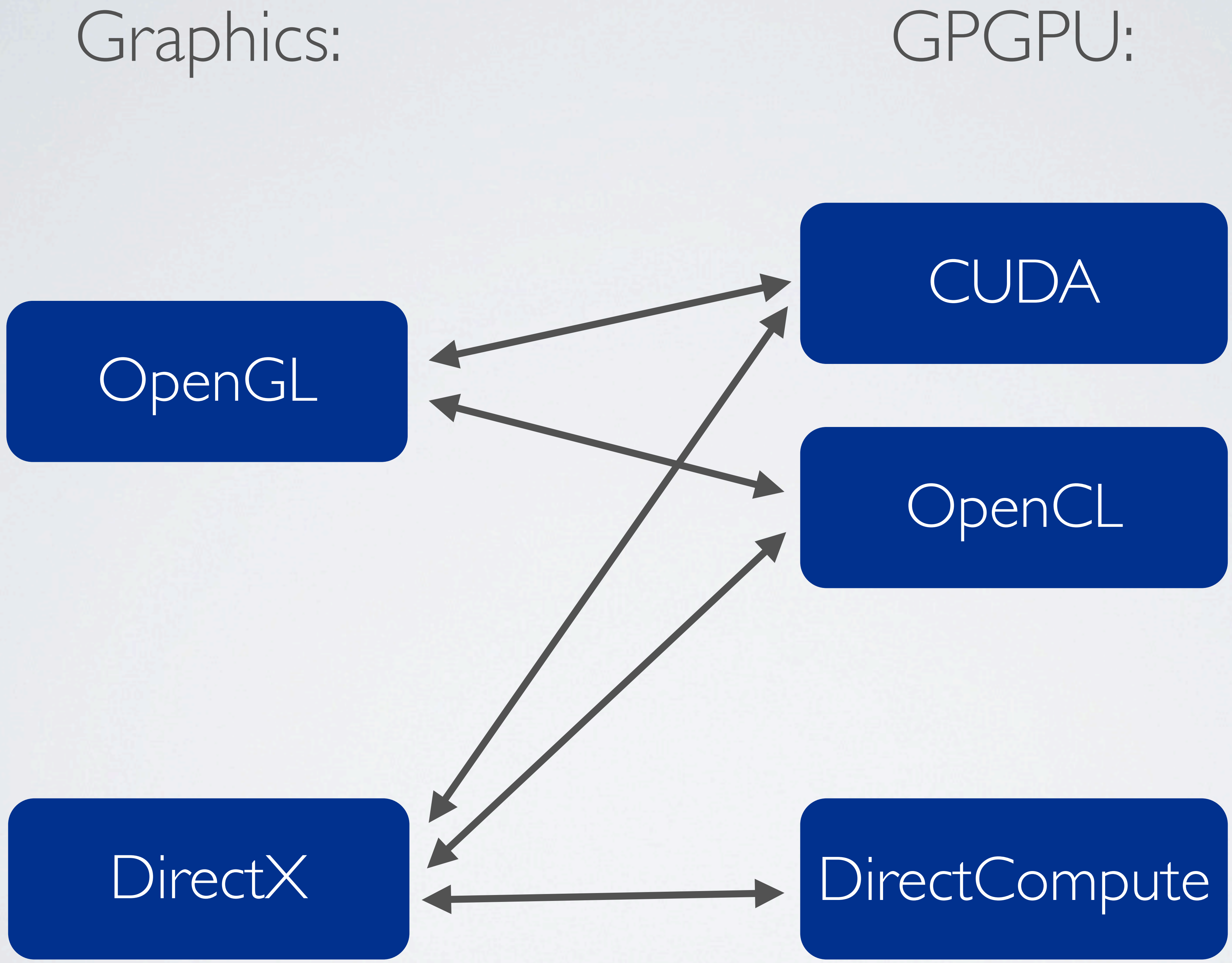
Advanced Computer Graphics 29.2.2012

Ville Timonen

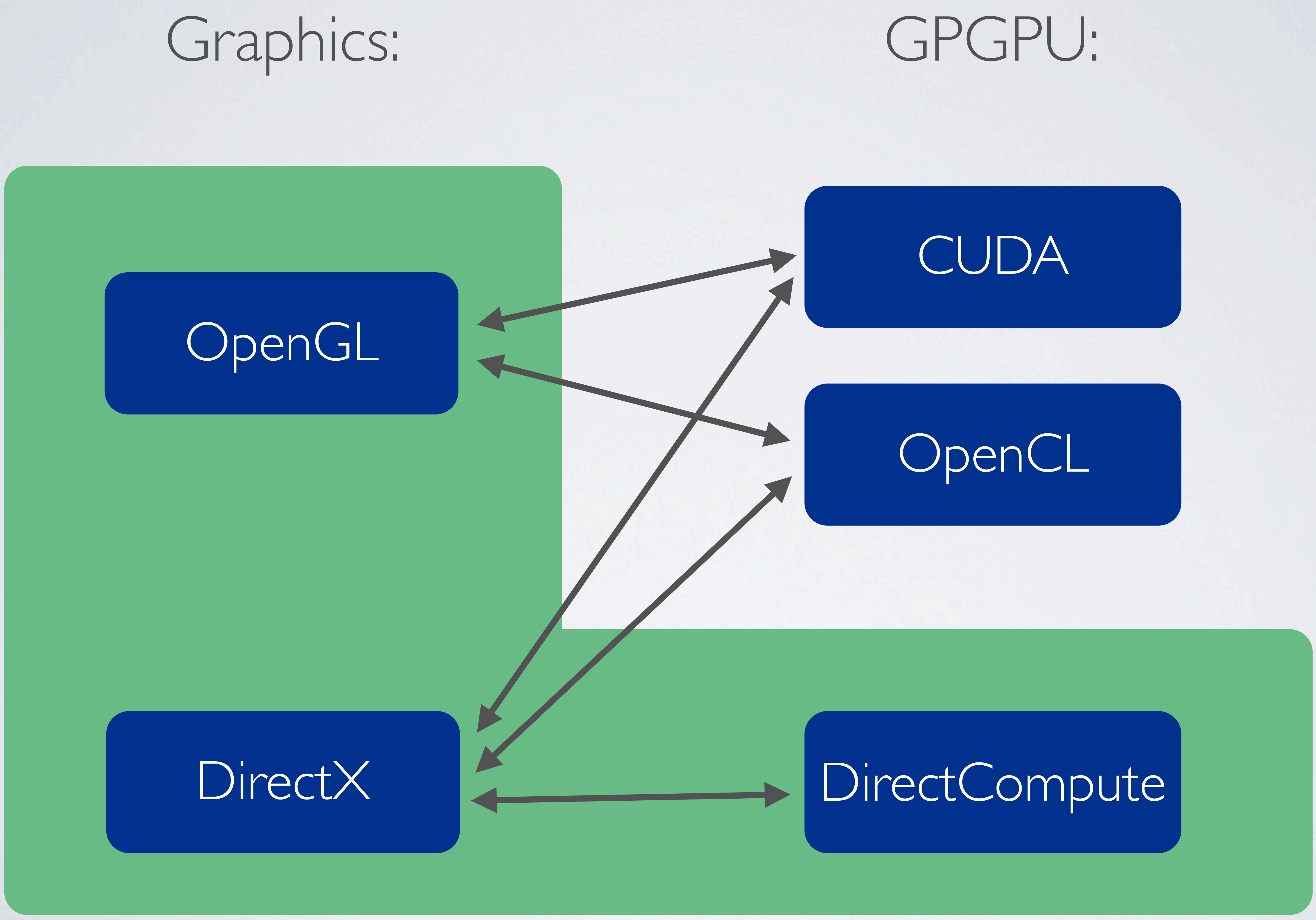
CONTENTS

- CUDA in the big picture
- When to use it in graphics apps
- How to use it
- Example: summed area tables (SAT)

CUDA IN THE BIG PICTURE



CUDA IN THE BIG PICTURE



WHEN TO USE CUDA

WHEN TO USE CUDA

- GPGPU is less limiting, allowing e.g.:
  - Arbitrary memory access patterns
  - On-chip memory communication
- CUDA when:
  - Only targeting NVidia hardware
  - Need advanced hardware features:
    - L1 config, vote functions, function pointers, etc

WHEN TO USE CUDA

- But choose it only when you really have to
- Don't underestimate optimized OpenGL operations
  - Driver writers know what they are doing
  - You will lose if you try to reinvent the wheel in CUDA
- For optimal performance, you need to know the target HW
  - If you didn't care for performance, you would do it in CPU

HOW TO USE CUDA

HOW TO USE CUDA

1 Initialize OpenGL

2 Initialize CUDA telling to share an OpenGL context

3 Pass data OpenGL -> CUDA

4 Perform calculations in CUDA

5 Pass results CUDA -> OpenGL

HOW TO USE CUDA

- Main data resources, sharable from OpenGL:
  - Linear allocations (OGL: buffers)
  - CUDA arrays (OGL: textures)
- Execution in kernels
  - Grouping into thread blocks
  - Results into linear allocations
    (can be copied into textures later on)

HOW TO USE CUDA

- You can choose either the runtime API or the driver API

Runtime API	Driver API
• Mix'n match GPU and CPU functions in same source files • Compile with nvcc into objects • Link into a complete binary *Pick me, I'm easy!*	• Compile GPU functions into PTX with nvcc • Compile CPU code separately (e.g. with gcc/g++) • Use CUDA as a normal library, upload the PTX file at runtime • Similar to uploading shader sources in OpenGL

EXAMPLE:
SUMMED AREA TABLES

SUMMED AREA TABLES

- Motivation: need to take an average over a region of pixels
  - Generation of texture mip-map levels
  - Fast blur filters (semi-glossy reflections, defocus blur)

SUMMED AREA TABLES

Each element s_{mn} of a summed-area table $\boldsymbol{S}$ contains the sum of all elements above and to the left of the original table/texture $\boldsymbol{T}$ [Crow84]

$$s_{mn} = \sum_{i=1}^{m} \sum_{j=1}^{n} t_{ij}$$

	1	2	3	4
1	2	3	2	1
2	3	0	1	2
3	1	3	1	0
4	1	4	2	2

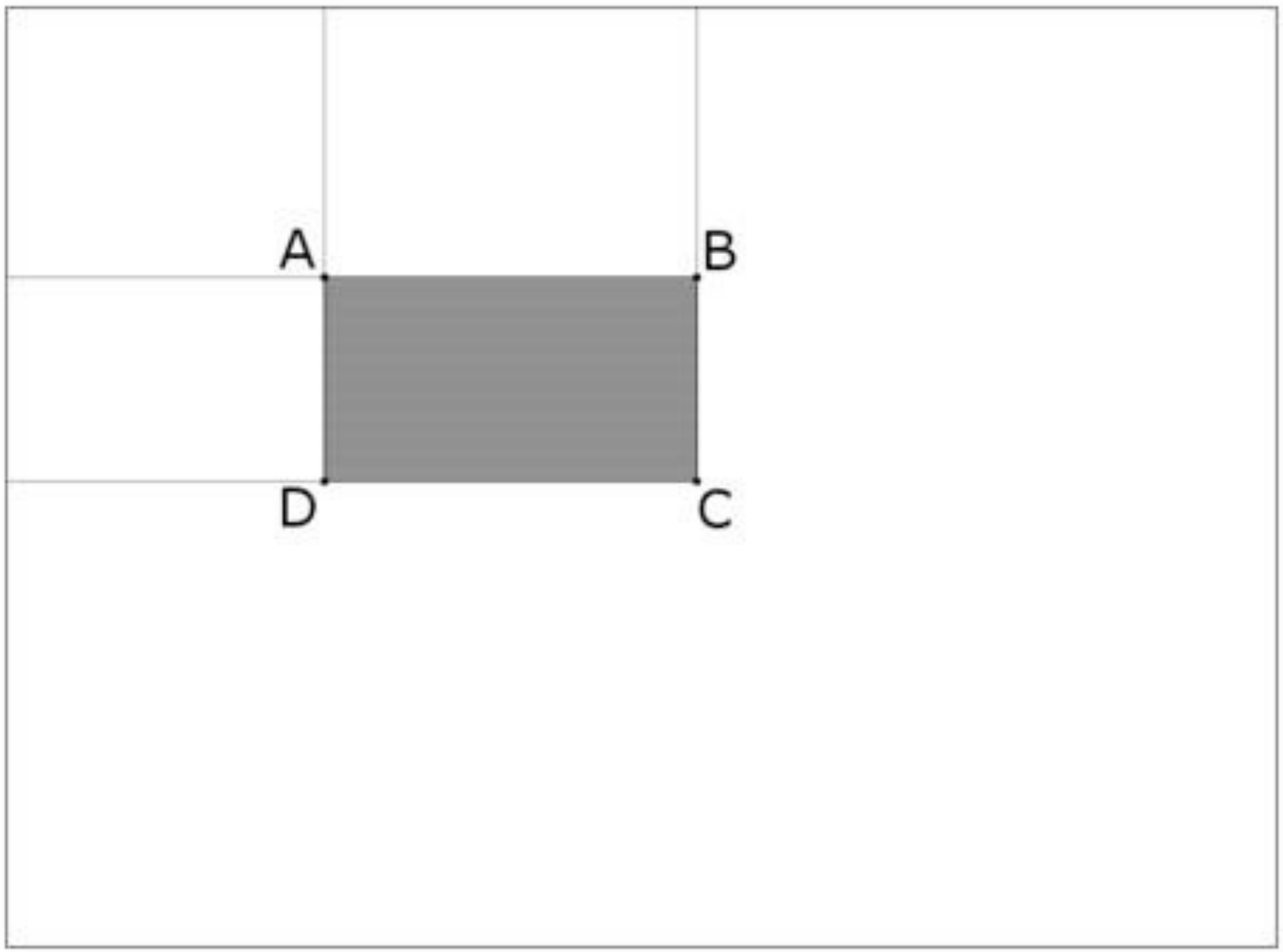
Original

2	5	7	8
5	8	11	14
6	12	16	19
7	17	23	28

Summed-area table

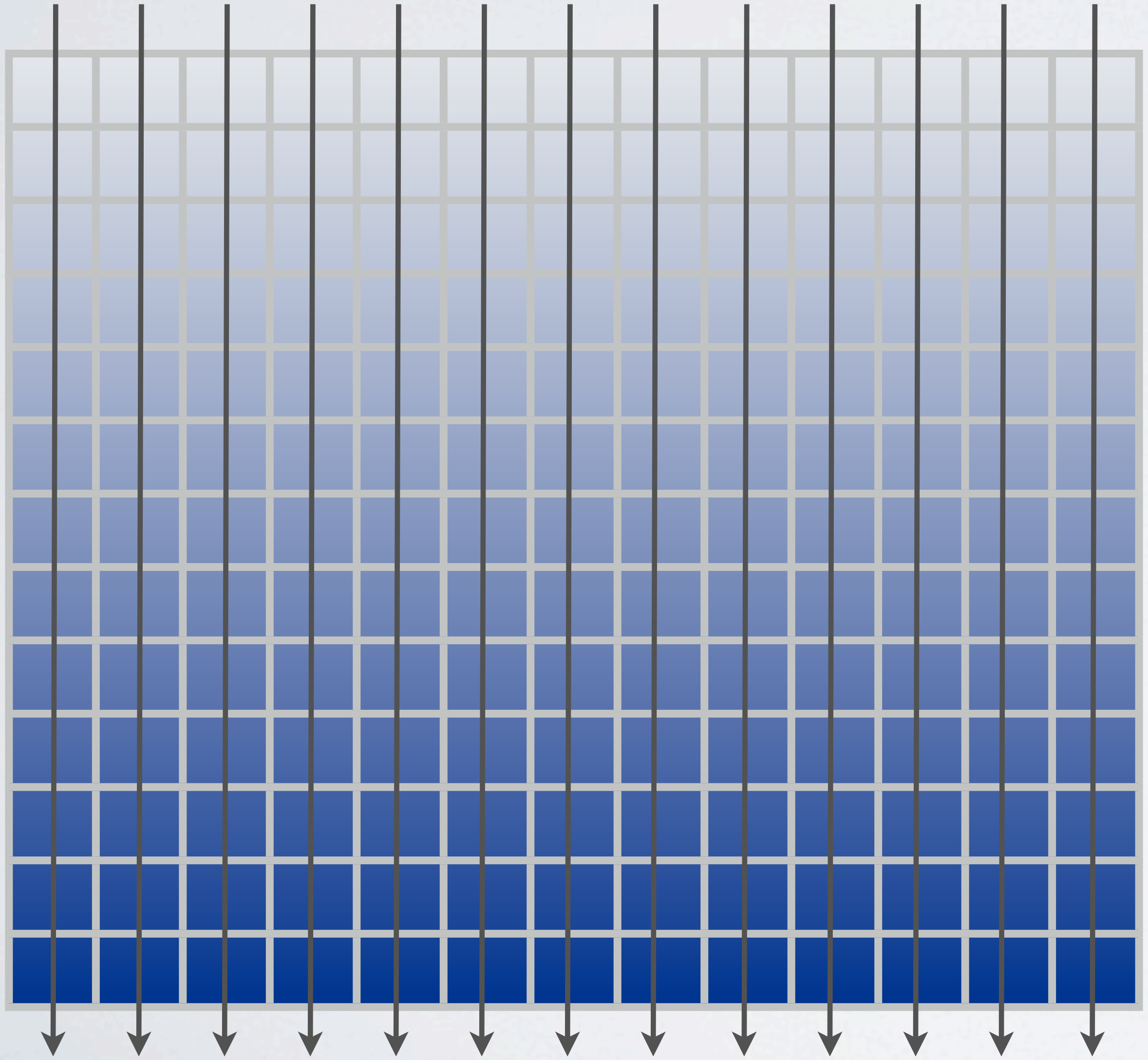
SUMMED AREA TABLES

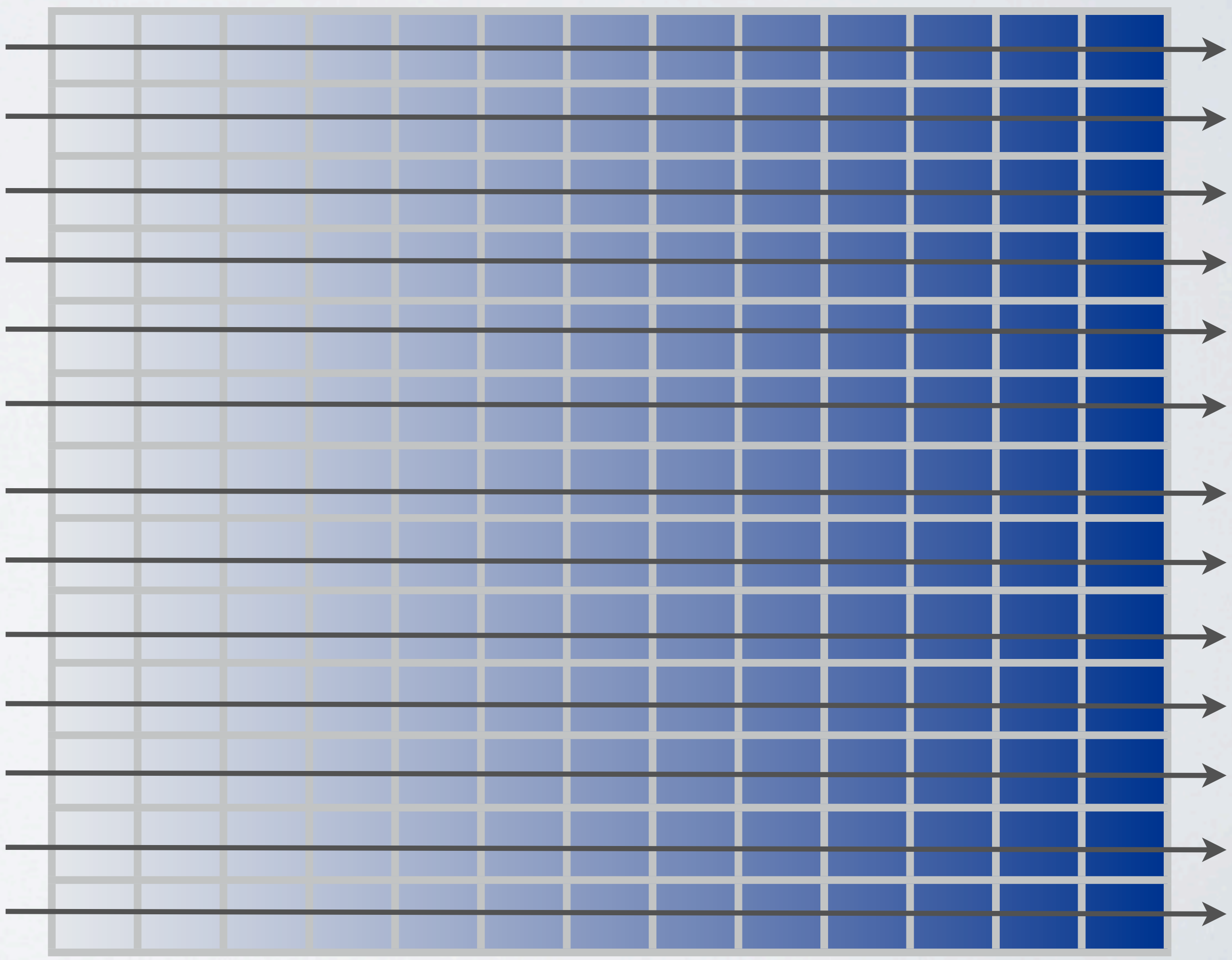
Sum of the region: c-b-d+a



SUMMED AREA TABLES

Vertical sweep + Horizontal sweep





SUMMED AREA TABLES: IMPLEMENTATION

SUMMED AREA TABLES

1. Initializations
2. Render a scene in OpenGL into a renderbuffer (via FBO)
3. Pass the renderbuffer + a result texture into CUDA
4. Perform sweeps in CUDA, write results in linear memory
5. Copy results into the result texture
6. Use the result texture in OpenGL in the usual fashion

SUMMED AREA TABLES

1. Initializations

2. Render a scene in OpenGL into a renderbuffer (via FBO)

```
createOpenGLWindow(...);
GLuint renBuf, satTex;

// Initialize CUDA to be used with OpenGL
cudaGLSetGLDevice(0);

// Register CUDA with the renderbuffer (renBuf)
struct cudaGraphicsResource* renBuf_cuda;
cudaGraphicsGLRegisterBuffer(&renBuf_cuda, renBuf,
        cudaGraphicsRegisterFlagsReadOnly); // CUDA will not write

// Register CUDA with the result texture (satTex)
struct cudaGraphicsResource* satTex_cuda;
cudaGraphicsGLRegisterImage(&satTex_cuda, satTex, GL_TEXTURE_2D,
        cudaGrsphicsRegisterFlagsWriteDiscard); // CUDA will overwrite, not read

// Allocate work data in CUDA (linear memory)
float4 *sweepData;
cudaMalloc(&sweepData, WIDTH*HEIGHT*sizeof(float4));
```

SUMMED AREA TABLES

1. Initializations
2. **Render a scene in OpenGL into a renderbuffer**
3. Pass the renderbuffer + a result texture into CUDA

SUMMED AREA TABLES

2. Render a scene in OpenGL into a renderbuffer (via FBO)

3. Pass the renderbuffer + a result texture into CUDA

4. Perform sweeps in CUDA, write results in linear memory

```
// Mapping tells OpenGL to flush changes and not to touch them until unmapped
cudaGraphicsMapResources(1, &renBuf_cuda);
cudaGraphicsMapResources(1, &satTex_cuda);

// Read in the CUDA pointers
float4 *renBufData; // Pointer to the buffer data
cudaGraphicsResourceGetMappedPointer(&renBufData,
        WIDTH*HEIGHT*sizeof(float4), renBuf_cuda);

struct cudaArray *satTexArray; // Cast the imported texture as CUDA array
cudaGraphicsSubResourceGetMappedArray(&satTexArray,
        satTex_cuda, 0, 0); // Tex ID (in case a cube map), mip-map level
```

SUMMED AREA TABLES

3. Pass the renderbuffer + a result texture into CUDA

4. Perform sweeps in CUDA, write results in linear m..

5. Copy results into the result texture

```
// In this example we ignore non-divisible-by-64 cases
sweepVert<<<WIDTH/64, 64>>>(renBufData, sweepData);
sweepHor <<<HEIGHT/64, 64>>>(sweepData);

__device__ void operator+=(float4 &a, const float4 b) {
    a.x += b.x; a.y += b.y; a.z += b.z; a.w += b.w;
}

__global__ sweepVert(const float4 *renBufData, float4 *sweepData) {
    unsigned int myId = blockIdx.x*blockDim.x + threadIdx.x;
    float4 mySum = make_float4(0.0f, 0.0f, 0.0f, 0.0f);

    for (int row = 0; row < HEIGHT; ++row) {
        mySum += renBufData[row*WIDTH + myId];
        sweepData[row*WIDTH + myId] = mySum;
    }
}

__global__ sweepHor(float4 *sweepData) {
    unsigned int myId = blockIdx.x*blockDim.x + threadIdx.x;
    float4 mySum = make_float4(0.0f, 0.0f, 0.0f, 0.0f);

    for (int col = 0; col < WIDTH; ++col) {
        mySum += sweepData[myId*WIDTH + col];
        sweepData[myId*WIDTH + col] = mySum;
    }
}
```

SUMMED AREA TABLES

1. Perform sweeps in CUDA, write results in linear memory

5. Copy results into the result texture

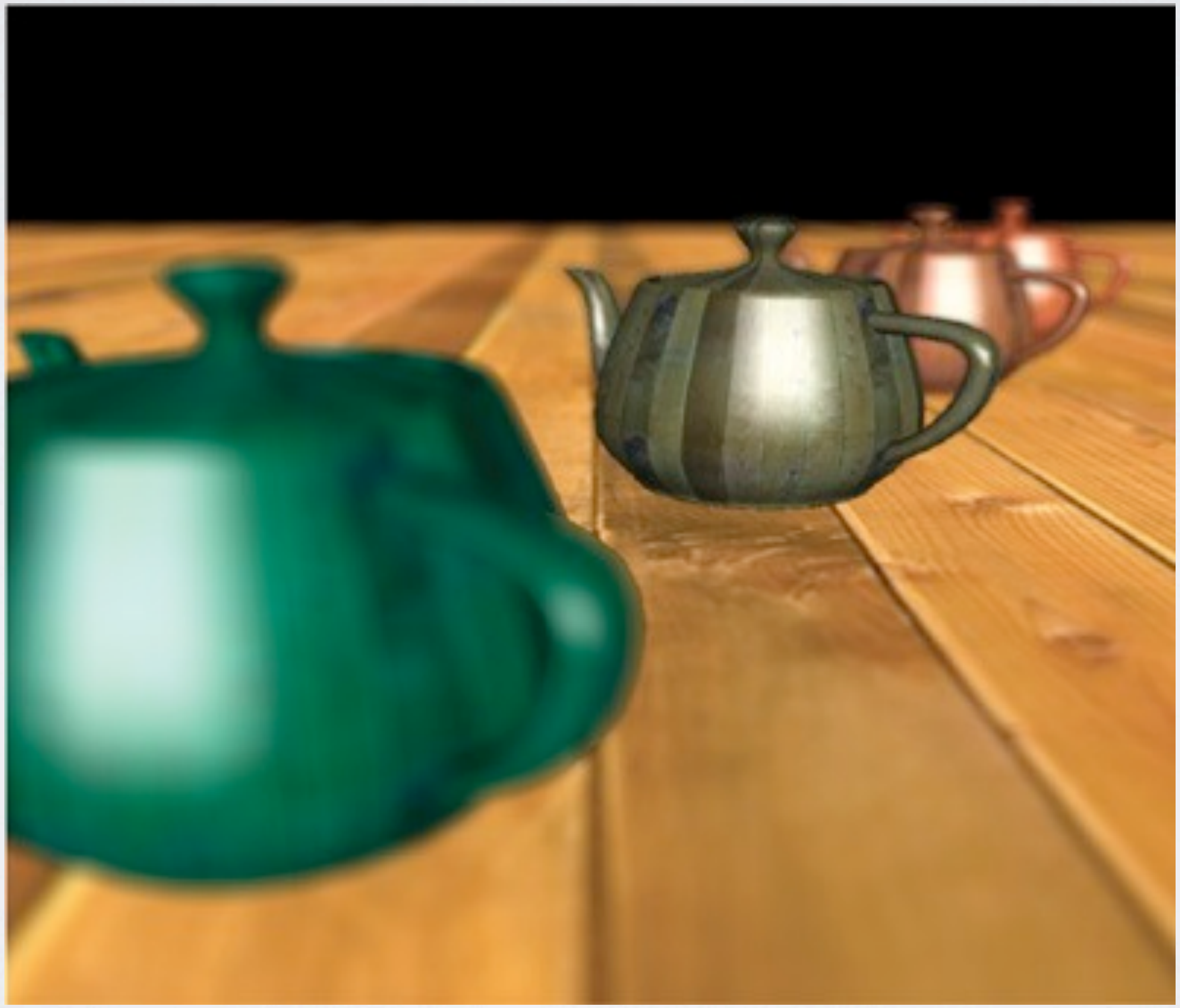
6. Use the result texture in OpenGL in the usual fashion

```
cudaMemcpyToArray(satTexArray, 0, 0, // wOffset, hOffset
        sweepData, WIDTH*HEIGHT*sizeof(float4),
        cudaMemcpyDeviceToDevice);

// Unmapping
cudaGraphicsUnmapResources(1, &satTex_cuda);
cudaGraphicsUnmapResources(1, &renBuf_cuda);

...

// At exit: unregister
cudaGraphicsUnregisterResource(satTex_cuda);
cudaGraphicsUnregisterResource(renBuf_cuda);
```

SUMMED AREA TABLES

5. Copy results into the result texture

6. **Use the result texture in OpenGL in the usual fash..**



QUESTIONS?