Heat Equation

Francesco Pasa
Enrico Panontin

Technische Universität München
Physics Department

Parallelisation of Physics Calculations on GPUs with CUDA

16 Juni 2016

Heat Equation

Our Code

Performances

OpenGL-CUDA interoperability

Demonstration

We consider heat propagation in a medium (e.g. a solid) far away from any state transition and we define:

ϵ : internal energy density

c : specific heat

ρ : mass density

T : temperature

By virtue of the first thermodinamic principle ($\Delta U = \Delta Q - L = \Delta Q$):

$$d\left[\int_{\Omega} c\rho T \,\mathrm{d}V\right] = \left[-\int_{\partial\Omega} \vec{J}\cdot\vec{n}\,\mathrm{d}S\right]\mathrm{d}t \tag{1}$$

Demonstration

$$d\left[\int_\Omega c\rho T \,\mathrm{d}V\right] = \left[-\int_{\partial\Omega} \vec{J}\cdot\vec{n}\,\mathrm{d}S\right]\mathrm{d}t$$

Newton-Fourier Law, heat spreads and temperature becomes uniform throughout the medium:

$$\vec{J} = -k\vec{\nabla}T \tag{2}$$

By substituting equation (2) in (1) and applying the *divergence theorem* one gets the **heat equation**:

$$\frac{\partial T}{\partial t} - \frac{k}{c\rho}\Delta T = 0 \tag{3}$$

Heat Equation

$$\frac{\partial T}{\partial t} - \frac{k}{c\rho}\Delta T = f(\vec{x}, t)$$

- First order in time
- Second order in space
- If we consider a heating system, we must add $f(\vec{x}, t)$

Numerical Solution

$$\frac{\partial T}{\partial t} - \frac{k}{c\rho}\Delta T = f(\vec{x}, t)$$

Discretize space and time: foreward difference for time, central difference for space:

$$T_{i,j}^{n+1} = T_{i,j}^{n} + \eta\left[T_{i+1,j}^{n} + T_{i-1,j}^{n} + T_{i,j+1}^{n} + T_{i,j-1}^{n} - 4T_{i,j}^{n}\right] \quad (4)$$

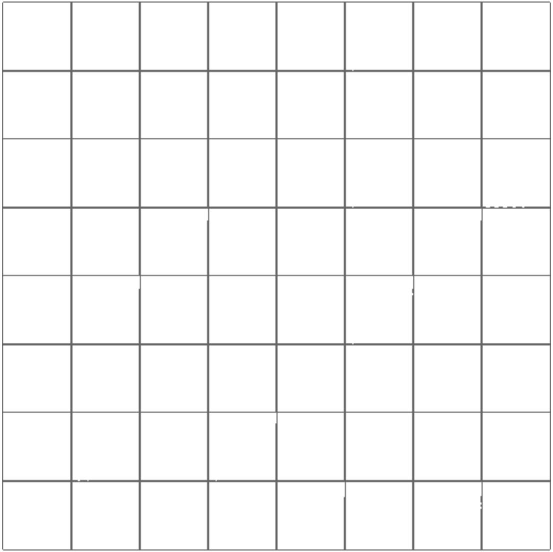
where n is the time index, i, j are x-index and y-index, $\eta = \frac{k\Delta t}{c\rho(\Delta x)^2}$.

General Structure

kjh

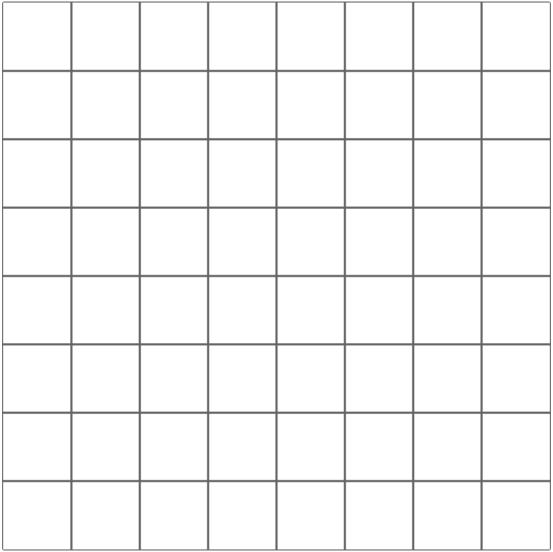
Shared Memory

Random errors



Shared Memory

Synchronized threads



Performances
GPU

descrizione della gpu usata (numero blocchi e thread eccecc)

Performances

Execution time vs blocks number

Performances

Execution time vs loops number

Performances

Execution time GPU vs CPU

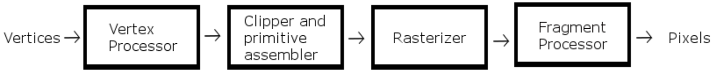
Performances
Different GPUs

Francesco Pasa
Enrico Panontin

OpenGL

OpenGL is a graphics API (similar in scope to Direct3D) that allows to use the graphics card to draw 2D and 3D vector graphics.

- 1992 - OpenGL is first released
- 2004 - OpenGL 2.0 is released. Adds support for programmable pipeline (shaders).
- 2010 - OpenGL 4.0 is released. Adds support for geometry shader (tesselation).



Old method for drawing

```
glColor3f(1.0f, 0.0f, 0.0f);
glBegin(GL_QUADS);
    glVertex2f(-0.25f, 0.25f);
    glVertex2f(-0.5f, -0.25f);
    glVertex2f(0.5f, -0.25f);
    glVertex2f(0.25f, 0.25f);
glEnd();
```

Moreover the API had to support many features, for example textures coordinates, lighting, shadows, coordinate transformation and perspective matrices.

This results in a complex API and lack of flexibility.

Modern approach

- Allow management of GPU memory from host CPU.
- Programmable pipeline with shaders.

Advantages: great flexibility, reduced complexity of API and driver implementations

Disadvantages: lots of boiler-plate code, not noob-friendly

OpenGL state machine

Since the rendering of the 3D scene is a very complex task, OpenGL uses the concept of state machine to simplify the API interface (avoid function with too many arguments).

```
// Tell OpenGL which array contains the data
glBindBuffer(GL_ARRAY_BUFFER, vbo);
// Specify how the data for position can be accessed
glVertexAttribPointer(0, size, GL_FLOAT, GL_FALSE, 0, 0);
// Enable the attribute
glEnableVertexAttribArray(0); // location = 0

// Draw
glDrawArrays(type, 0, vertex_num);
```

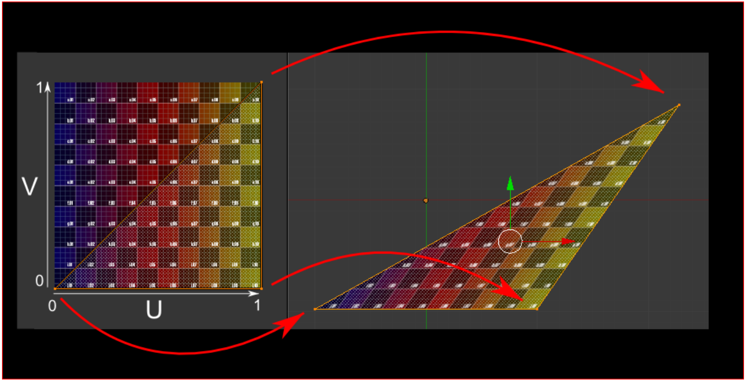
OpenGL VBOs

VBO (Vertex Buffer Object) - is an array of data in the GPU memory for storing vertices.

```
// Tell OpenGL we want to allocate array
glGenBuffers(1, &vbo);
/* Tell OpenGL we want to modify the state
    of the following array */
glBindBuffer(GL_ARRAY_BUFFER, vbo);

// Actually allocate memory
glBufferData(GL_ARRAY_BUFFER, size,
    NULL, GL_DYNAMIC_DRAW);

// Tell OpenGl we are done with the array
glBindBuffer(GL_ARRAY_BUFFER, 0);
```

OpenGL Textures

Texture - is an image that is mapped to vertices.



OpenGL Textures

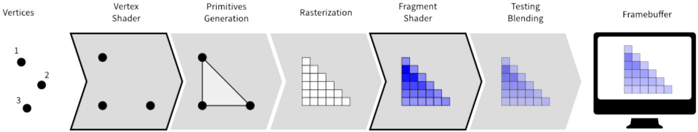
```
// Tell OpenGl we want to allocate a texture
glGenTextures(1, &texture);
/* Tell OpenGl we are going to modify the state
    of the following texture */
glBindTexture(GL_TEXTURE_2D, texture);

// Set interpolation method to linear
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

// Actually allocate memory
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, width, height,
    0, GL_RGBA, GL_UNSIGNED_BYTE, NULL);

// Tell OpenGl we finished modifying this etxture
glBindTexture(GL_TEXTURE_2D, 0);
```

OpenGL shaders

Shaders → programs that run on the GPU (analogous to CUDA kernels).

Three types:

- Vertex shader - transforms vertex coordinates.
- Fragment shader - transforms pixel colors.
- Geometry shader - can add vertices.



Vertex shader

```
#version 330

layout(location = 0) in vec2 position;
out vec2 texCoord;

void main() {
    texCoord = vec2(position.x/2 + 0.5, position.y/2 + 0.5);
    gl_Position = vec4(position.x, position.y, 1.0, 1.0);
}
```

Fragment shader

```
#version 330

uniform sampler2D tex;
in vec2 texCoord;
out vec4 colorOut;

void main() {
    float strength = texture(tex, texCoord).x;

    // Colormap
    //   1     red      (1, 0, 0)
    //   0.75  yellow   (1, 1, 0)
    //   0.5   green    (0, 1, 0)
    //   0.25  cyan     (0, 1, 1)
    //   0     blue     (0, 0, 1)
    // RED
    float red = (strength > 0.75) ?
        1 : ((strength > 0.5) ?
            (strength - 0.5) * 4 : 0);
    // GREEN
    float green = (strength <= 0.75 && strength > 0.25) ?
        1 : ((strength > 0.75) ? 1 - strength : strength) * 4;
    // BLUE
    float blue = (strength > 0.5) ?
        0 : ((strength > 0.25) ?
            (0.5 - strength) * 4 : 1);

    colorOut = vec4(red, green, blue, 1);
}
```

How to draw

```
// Select texture unit
glActiveTexture(GL_TEXTURE0);
// Use this texture
glBindTexture(GL_TEXTURE_2D, texture);

// Assign texture unit index to the fragment shader
glUniform1i(1, 0); // location = 1, texture unit = 0

/*** We need a support to draw our texture ***/
// Tell OpenGL which array contains the data
glBindBuffer(GL_ARRAY_BUFFER, vbo);
// Specify how the data for position can be accessed
glVertexAttribPointer(0, 2, GL_FLOAT, GL_FALSE, 0, NULL);
// Enable the attribute
glEnableVertexAttribArray(0); // location = 0

// Draw
glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);

// Disable array and texture
glBindBuffer(GL_ARRAY_BUFFER, 0);
glBindTexture(GL_TEXTURE_2D, 0);
```

OpenGL-CUDA interperability

OpenGL-CUDA interoperability

Execution model

Registering and mapping