Pixel Power (P2)

Eliane Kabkab – System Integrator
Nageswar Keetha – Language Guru
Eric Liu – Project Manager
Kaushik Viswanathan – System Architect
Today’s Landscape

- Multi-core computers
- Increasing importance of GPU not only for graphics but for general computing
- Stream processing paradigm – apply the same function to each element in a data set at the same time
Goals

• Programming using stream processing
• Eliminates some synchronization-related issues with parallel processing
• Abstract away details of thread management
• Code that can run on GPU’s
• Familiarity – programmers comfortable with C will find it easier to learn and use
Applications

• Large data sets such as scientific computation and high performance computing
• Image manipulation and processing
• 3D graphics and rendering (shaders)
• Parallelization of existing C code
Features

- Imperative, statically typed
- Stream functions – automatically split function into multiple threads
- Shader functions – running code on GPU’s
- New types: [type-name]NxM
  - Image2D, matrix, float2x3
- Built-in functions: manipulation of newly defined types
Syntax: C + HLSL + custom - reserved
<table>
<thead>
<tr>
<th>Reserved</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>int</td>
<td>switch</td>
<td>break</td>
<td>continue</td>
<td></td>
<td></td>
</tr>
<tr>
<td>uint</td>
<td>case</td>
<td>if</td>
<td>return</td>
<td></td>
<td></td>
</tr>
<tr>
<td>stream</td>
<td>default</td>
<td>else</td>
<td></td>
<td>Matrix</td>
<td></td>
</tr>
<tr>
<td>float</td>
<td>Image2d</td>
<td>do</td>
<td></td>
<td>sizeof</td>
<td></td>
</tr>
<tr>
<td>TextBox</td>
<td>Image3d</td>
<td>while</td>
<td></td>
<td>typedef</td>
<td></td>
</tr>
<tr>
<td>Vector</td>
<td>Texture2d</td>
<td>for</td>
<td></td>
<td>extern</td>
<td></td>
</tr>
<tr>
<td>char</td>
<td>Quad</td>
<td>struct</td>
<td></td>
<td>SHelper</td>
<td></td>
</tr>
<tr>
<td>String</td>
<td>static</td>
<td>enum</td>
<td></td>
<td>shader</td>
<td></td>
</tr>
</tbody>
</table>
Language constructs

- Stream
- Shared buffer for stream functions (mapping to a list of threads)
- Shader
- Vector, Matrix, Tensor
- Easy declaration – swizzling
- Example: color = color.rrg
## Inbuilt types

<table>
<thead>
<tr>
<th>Intrinsic Type</th>
<th>Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalar</td>
<td>One-component scalar (1D)</td>
</tr>
<tr>
<td>Vector, Matrix</td>
<td>Multiple-component tensors (1D, 2D)</td>
</tr>
<tr>
<td>Image</td>
<td>Image object (2D, 3D)</td>
</tr>
<tr>
<td>Texture</td>
<td>Texture object (2D)</td>
</tr>
<tr>
<td>TextBox</td>
<td>TextBox object</td>
</tr>
<tr>
<td>Struct</td>
<td>Custom structure</td>
</tr>
<tr>
<td>Svar</td>
<td>Shader variable list</td>
</tr>
<tr>
<td>Enum</td>
<td>C-style enumerations</td>
</tr>
<tr>
<td>User Defined</td>
<td>typedef</td>
</tr>
</tbody>
</table>
Inbuilt functions

**TEXTURE**

- `int Create(Texture* tex, uint2 resolution, PIXEL_FORMAT format, TEX_ACCESS access)`
- `Image2D* GetImage(Texture* tex)`
- `int SetRenderTarget(Texture* tex)`
- `int Clear(Texture* tex, RGBA color)`
- `int Release(Texture* tex)`

**TextBox**

- `int CreateFont (TextBox* tBox, float2 size, bool isBold, bool isItalic, String fontType)`
- `int Draw(TextBox* tBox, RGBA color, Rect rect, TEXT_ALIGN alignment)`

**C-GLOBALS**

- `void AddElementTo([User-defined-type]* container, float2 position, [Type]* value)`
- `int BindSvar(svar* SVar, string address, [Type]* value)`
- `int SetShader(shader* Name)`

**STREAM SYNCHRONIZATION**

- `void synch()`
Inbuilt states

- Timing information – between states
- BackBuffer – screen
- Input – mouse/keyboard info
- SHADER_IN – pixel information
- STREAM_IN – current thread
Program Structure

- Multiple files allowed like C
- `int init()` – Once per program
- `int main()` – Once per frame (Required)
- `int input_callback(Input*)` – Once per keyboard / mouse event
//Converts an image to black and white

SHelper float calcRGB(int r, int g, int b) {
    //Computation for rgbColor.rgb
    return r * 0.3 + g * 0.6 + b * 0.1;
}

shader RGBToBW {
    //Read the color of the current pixel
    float4 rgbColor = Tex2D(&rgbIn, PS_IN.texCoord, TEX_FILTER_LINEAR);
    rgbColor.rgb = calcRGB(rgbColor.r, rgbColor.g, rgbColor.b);
    return rgbColor;
}

//Shader variable list
svarRGBToBW
{
    Image rgbIn;
};

//Main render loop
int main() {
    Image img;
    img.Load("example.jpg");
    //Target texture for the shader
    SetRenderTarget(&backBuffer);
    BindSvar("RGBToBW", "rgbin", &img);
    SetShader("RGBToBW");
    //Draw the default Quad to the screen using
    //shader RGBToBW
    screenAligned.Draw();
    return 0;
}
.p2 & .h files

Cleanup

Exclude C++ + HLSL – p2

File Split (Scoping)

Lex + Parse Custom

List of .cpp, .h, and .fx files

SDT

Custom Code Injection

VC++ compiler + linker

HLSL compiler

.exe
Implementation

• DirectX based mini-engine, Windows .exe, Visual C++ compiler
• [.p2, .h] -> [.cpp, .h, .fx] + graphics and threading API -> .exe
Tools and API’s used

- Visual C++ express IDE
- Visual C++ command line compiler
- Flex/Bison
- Assembla / Tortoise for SVN
- DirectX 9 SDK
- Custom graphics API
- Google Groups
Testing methodology

Gradual Step-by-Step Testing

• Unit testing
  Design relatively independent sub-units
  Test shared structures
  Test each sub-unit extensively

• Integration testing
  Test shared interaction

• System Testing
  Test whole compiler
Challenges

Technical Challenges

• New to Image Processing and Graphics Programming
• New APIs to work on

Human Challenges

• Diverse backgrounds
• Scheduling
However,

*Not only did we*

- Come up with a comprehensive language
- Manage to compile code and run it with visualizable results

*Our language*

- Is easy to learn for a C/C++ programmer
- Is easier than already existing graphics oriented languages
- Does the work for you and saves you a lot of time