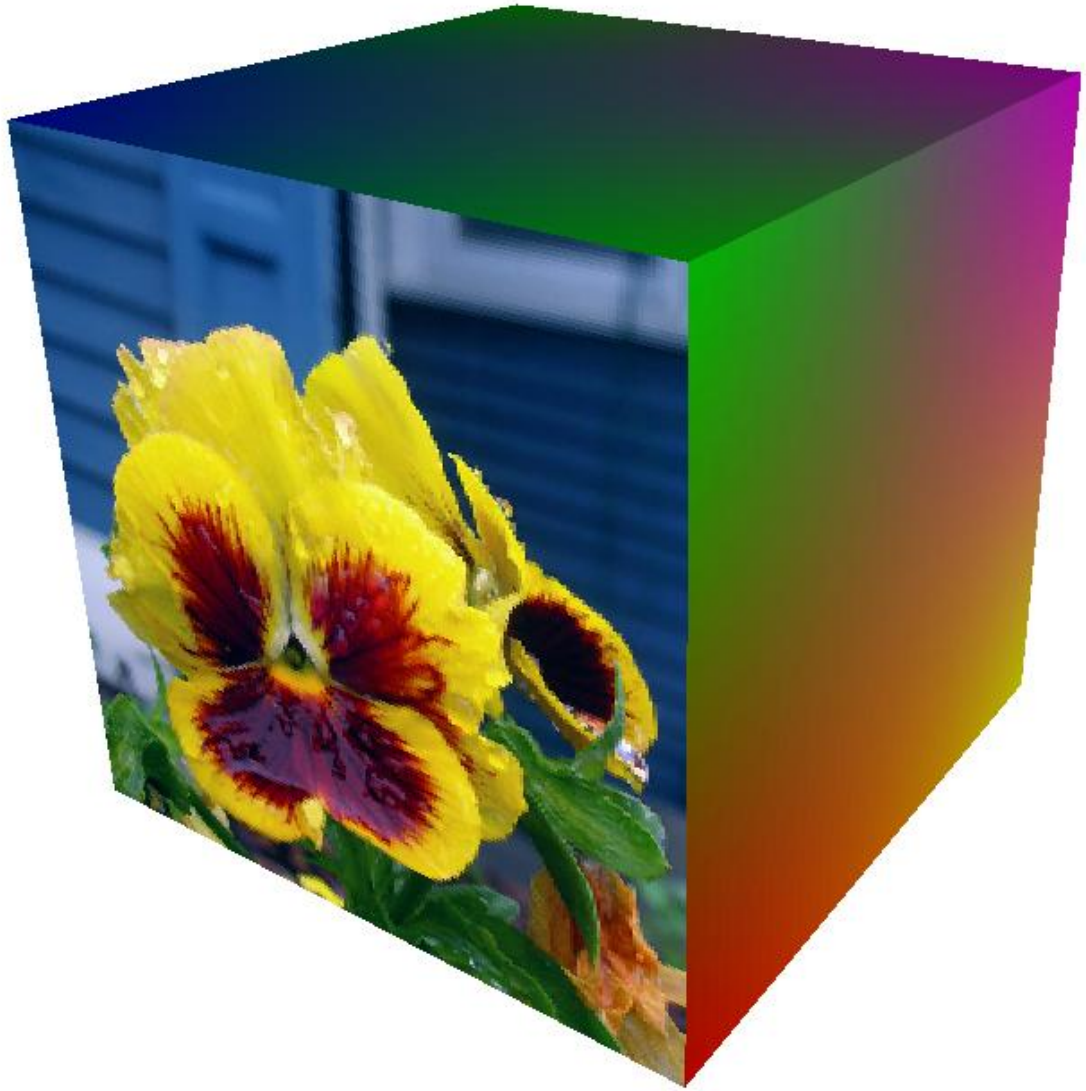
Pixel Bender – Building a Domain Specific Language on the GPU

Bob Archer

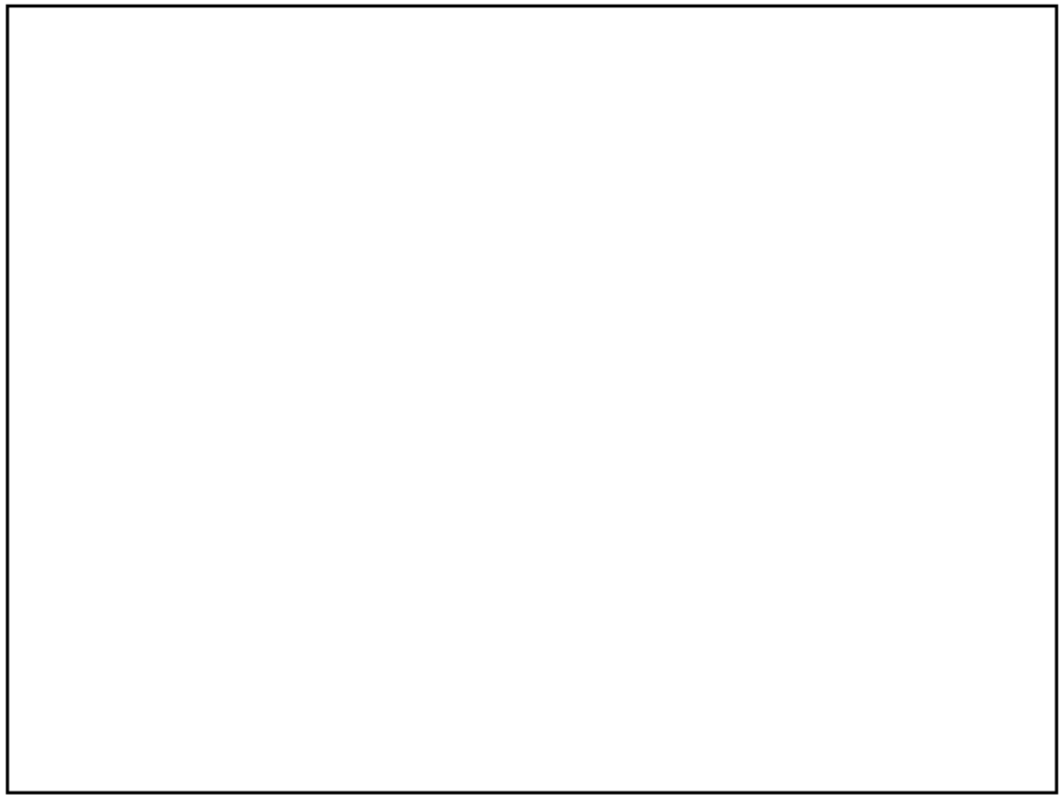
Demo

- Twirl
- Radial blur
- Oil paint filter
- Droste
- Fade to black
- Mandelbulb
- Ray tracer

What’s happening?



What’s happening?



What’s happening?

What’s happening?

What’s happening?

What's happening?

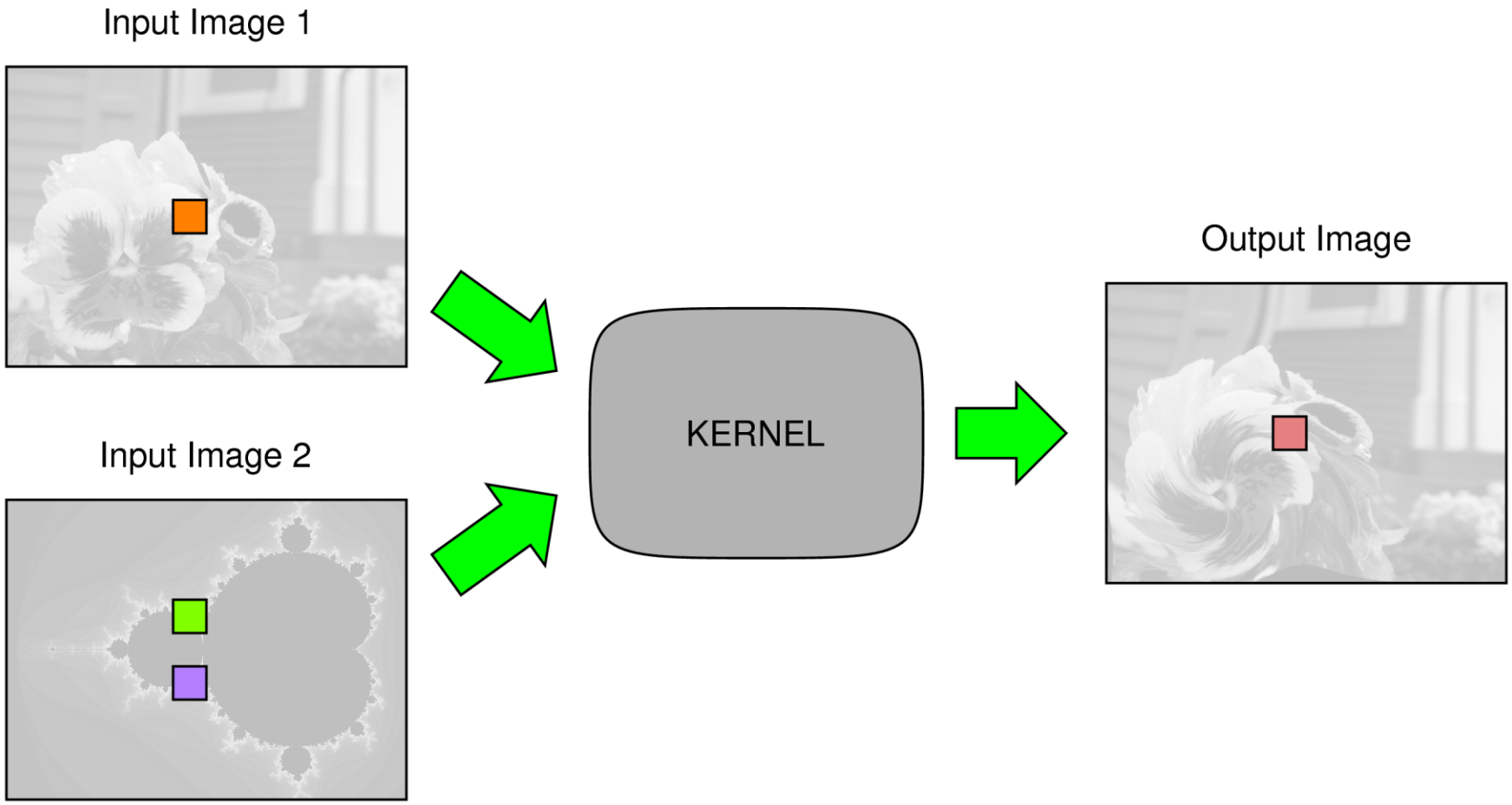
- 3D system – OpenGL + OpenGL Shading Language
- Ignore all of the vertex processing features
- Draw one single polygon – rectangular, the size of the desired output
- Run a fragment shader that computes the desired output
- Fragments become square or rectangular pixels

Write a function that produces a single output fragment

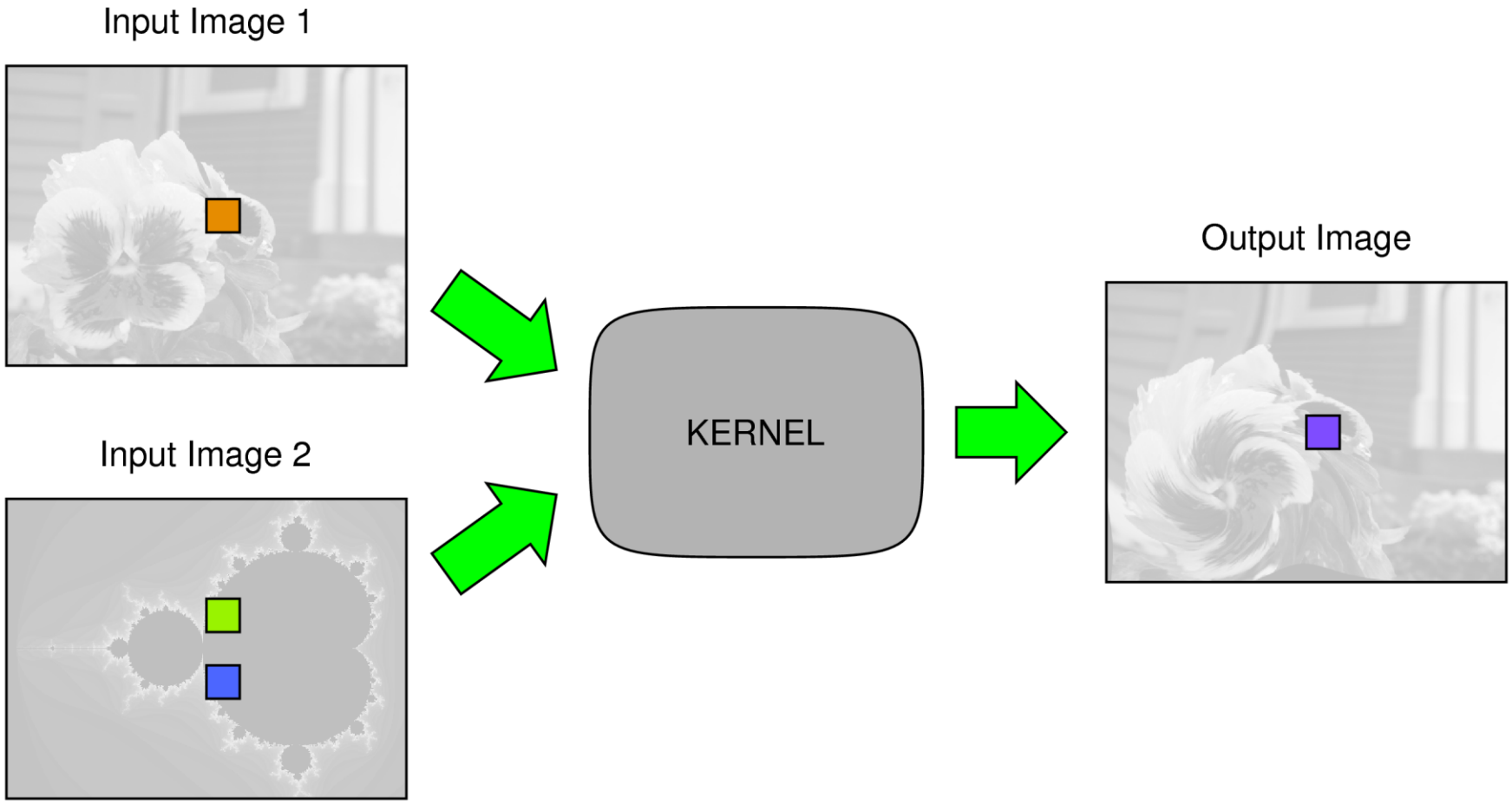
Pixel Bender programming model

Write a function that produces a single output ~~fragment~~ pixel

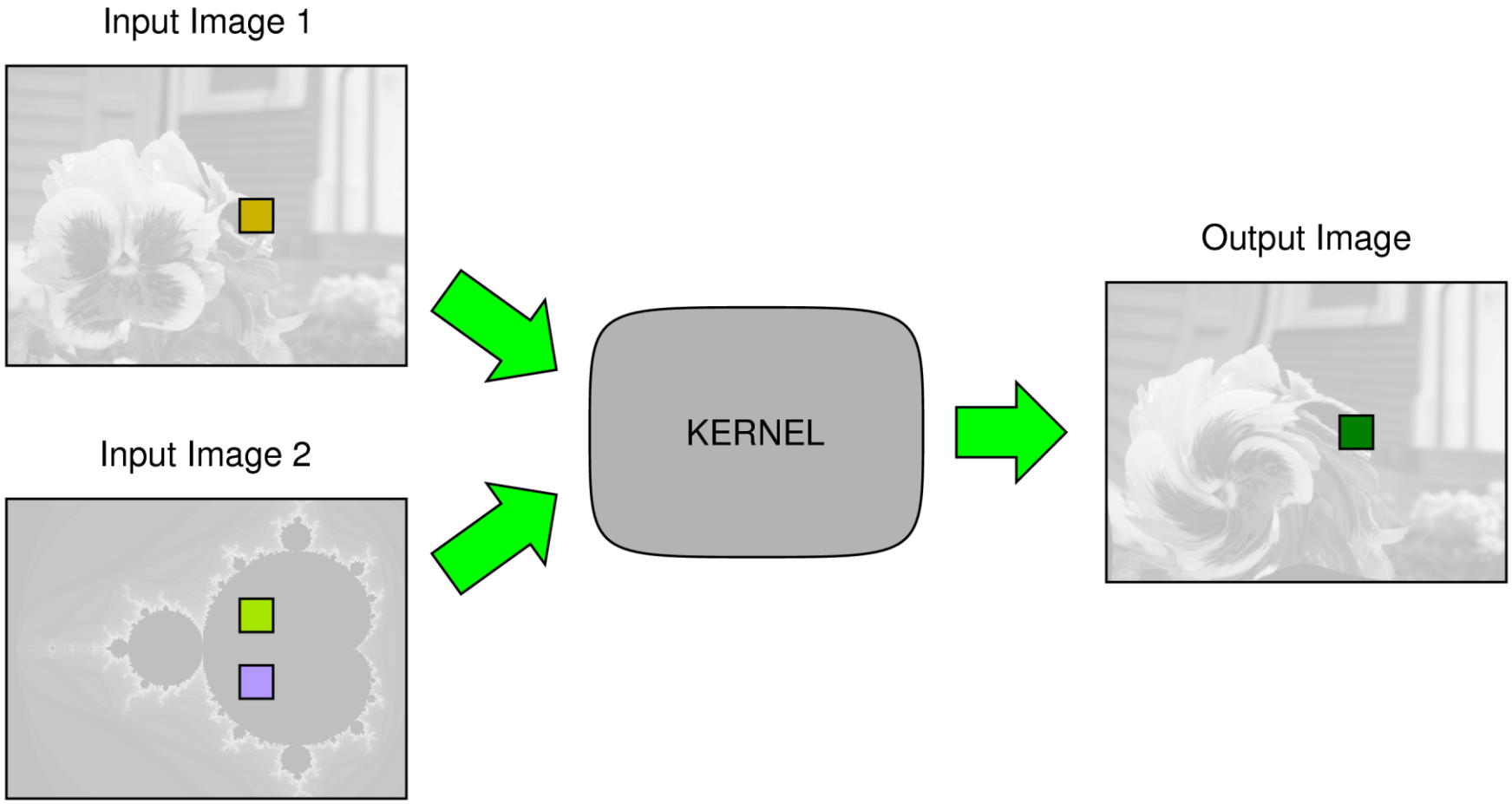
Pixel Bender programming model



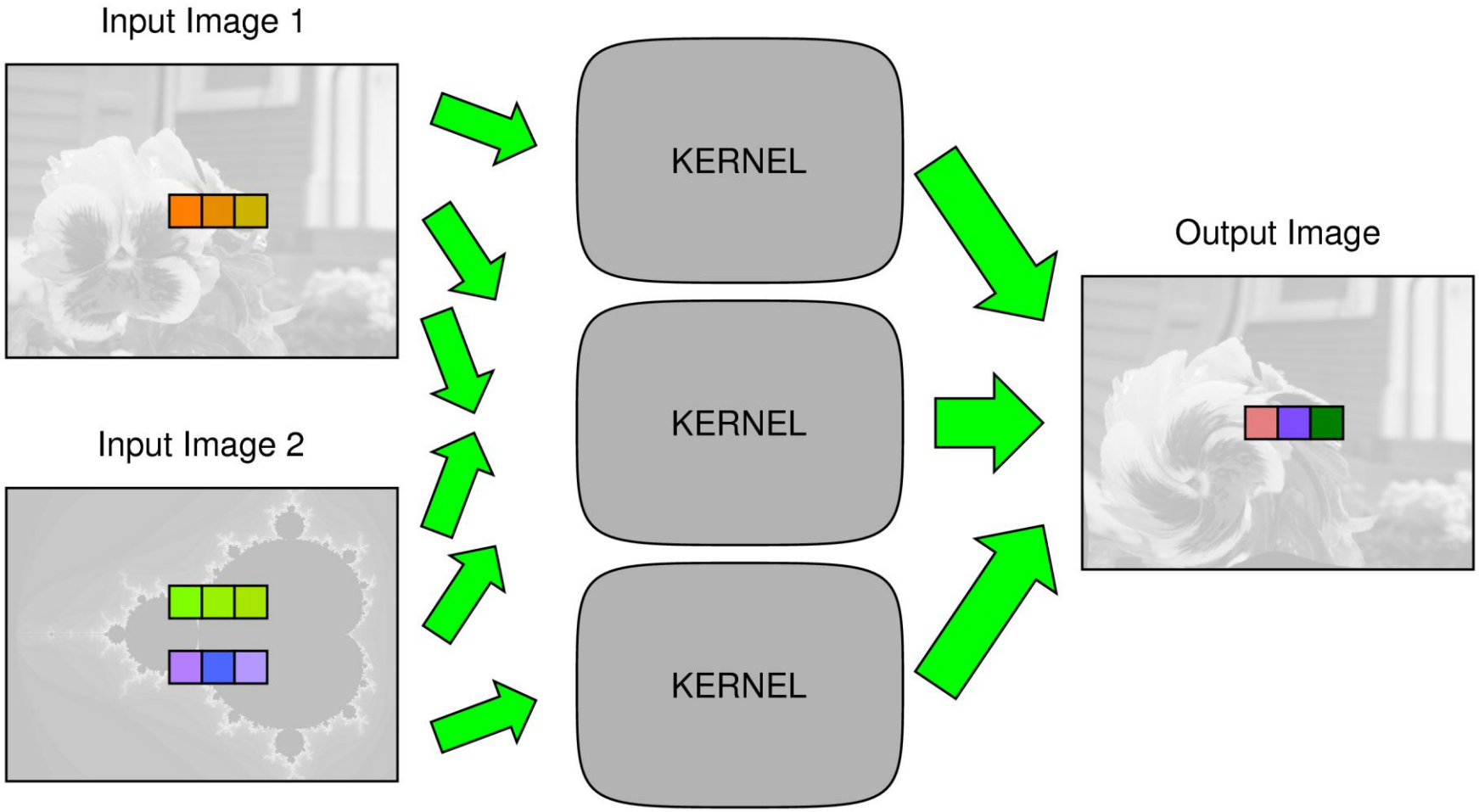
Pixel Bender programming model



Pixel Bender programming model



Pixel Bender programming model



Code walkthrough

```
<languageVersion : 1.0;>
kernel FadeToBlack
<   namespace : "AIF test";
    vendor : "Adobe";
    version : 1;
    description : "Fade out the image";
>
{
    parameter float fade
    <
        minValue : 0.0;
        maxValue: 1.0;
        defaultValue : 1.0;
    >;

    input image4 src;
    output pixel4 dst;

    void
    evaluatePixel()
    {
        dst = sampleNearest(src,outCoord());
        dst.rgb *= fade;
    }
}
```

Code walkthrough

```
kernel FadeToBlack
{
    parameter float fade;
    input image4 src;
    output pixel4 dst;

    void evaluatePixel()
    {
        dst = sampleNearest( src, outCoord() );
        dst.rgb *= fade;
    }
}
```

Code walkthrough

```
kernel FadeToBlack
{
    parameter float fade;
    input image4 src;
    output pixel4 dst;

    void evaluatePixel()
    {
        dst = sampleNearest( src, outCoord() );
        dst.rgb *= fade;
    }
}
```

Code walkthrough

```
kernel FadeToBlack
{
    parameter float fade;
    input image4 src;
    output pixel4 dst;

    void evaluatePixel()
    {
        dst = sampleNearest( src, outCoord() );
        dst.rgb *= fade;
    }
}
```

Code walkthrough

```
kernel FadeToBlack
{
    parameter float fade;
    input image4 src;
    output pixel4 dst;

    void evaluatePixel()
    {
        dst = sampleNearest( src, outCoord() );
        dst.rgb *= fade;
    }
}
```

Code walkthrough

```
kernel FadeToBlack
{
    parameter float fade;
    input image4 src;
    output pixel4 dst;

    void evaluatePixel()
    {
        dst = sampleNearest( src, outCoord() );
        dst.rgb *= fade;
    }
}
```

Code walkthrough

```
kernel FadeToBlack
{
    parameter float fade;
    input image4 src;
    output pixel4 dst;

    void evaluatePixel()
    {
        dst = sampleNearest( src, outCoord() );
        dst.rgb *= fade;
    }
}
```

Code walkthrough

```
kernel FadeToBlack
{
    parameter float fade;
    input image4 src;
    output pixel4 dst;

    void evaluatePixel()
    {
        dst = sampleNearest( src, outCoord() );
        dst.rgb *= fade;
    }
}
```

Code walkthrough

```
kernel FadeToBlack
{
    parameter float fade;
    input image4 src;
    output pixel4 dst;

    void evaluatePixel()
    {
        dst = sampleNearest( src, outCoord() );
        dst.rgb *= fade;
    }
}
```

Code walkthrough

```
kernel FadeToBlack
{
    parameter float fade;
    input image4 src;
    output pixel4 dst;

    void evaluatePixel()
    {
        dst = sampleNearest( src, outCoord() );
        dst.rgb *= fade;
    }
}
```

Code walkthrough

```
kernel FadeToBlack
{
    parameter float fade;
    input image4 src;
    output pixel4 dst;

    void evaluatePixel()
    {
        dst = sampleNearest( src, outCoord() );
        dst.rgb *= fade;
    }
}
```

GLSL version of fade to black filter

```
#define _PROFILE_TILED
#define _PROFILE_NORMALIZED

uniform vec2 _outOffset;
uniform vec2 _outPixelExtent;
uniform float fade;
uniform sampler2D src_sampler;
uniform vec2 src_offset;
uniform vec2 src_delta;

vec4 temp_dst;

void main(void)
{
    temp_dst = texture2D( src_sampler,
      ( gl_TexCoord[0].xy * _outPixelExtent + _outOffset - src_offset ) * src_delta );
    temp_dst.rgb *= fade;
    gl_FragData[0] = temp_dst;
}
```

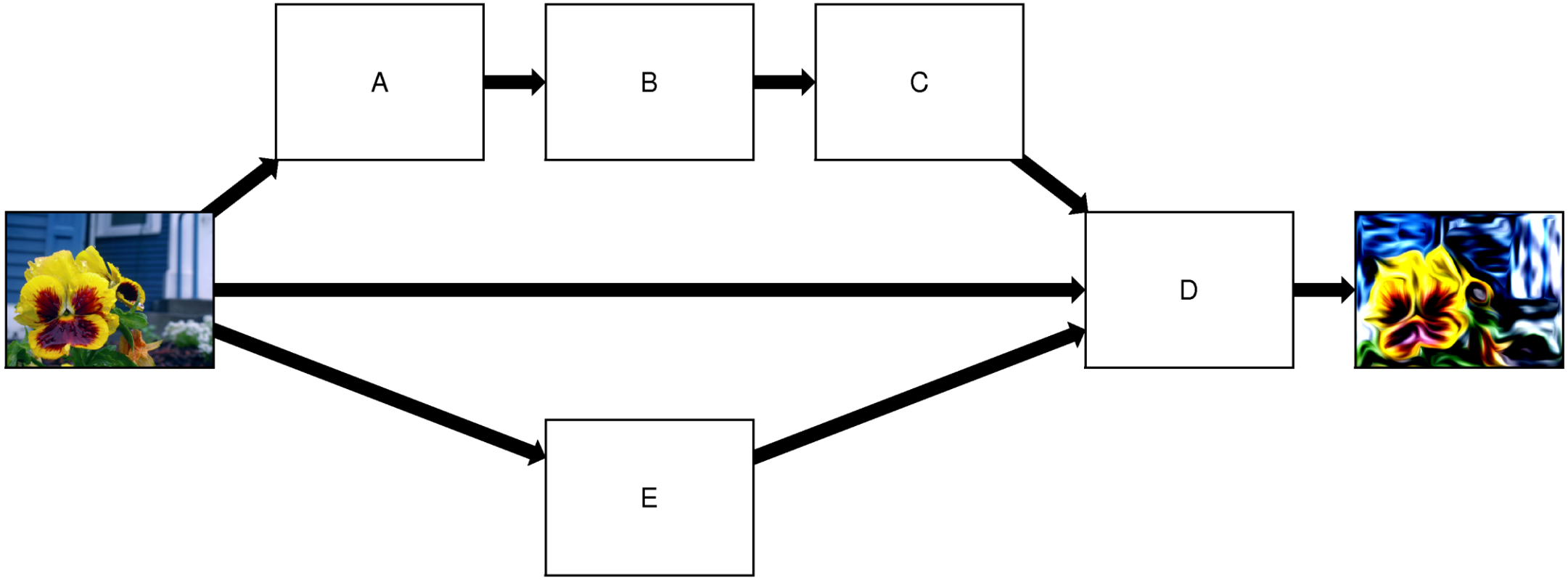
GLSL version of fade to black filter

```
#define _PROFILE_TILED
#define _PROFILE_NORMALIZED

uniform vec2 _outOffset;
uniform vec2 _outPixelExtent;
uniform float fade;
uniform sampler2D src_sampler;
uniform vec2 src_offset;
uniform vec2 src_delta;

vec4 temp_dst;

void main(void)
{
    temp_dst = texture2D( src_sampler,
      ( gl_TexCoord[0].xy * _outPixelExtent + _outOffset - src_offset ) * src_delta );
    temp_dst.rgb *= fade;
    gl_FragData[0] = temp_dst;
}
```

Pixel Bender

- At first glance, it looks a lot like GLSL, but…
  - Has many additions tuned towards image processing
    - Non-square pixels
    - Image offsets
    - Graphs
    - Per-frame operations (as opposed to per-pixel)
    - Region reasoning
  - Remove some of the 3D boilerplate (but add 2D boilerplate)

Graphs



Per-frame operations - evaluateDependents

```
kernel Twirl
{
    parameter float radius_in;
    parameter float2 center_in;
    parameter float twirl_angle_in;

    dependent float radius;
    dependent float2 center;
    dependent float twirl_angle;

    void evaluateDependents()
    {
        radius = radius_in * 256.0;
        center = center_in * 512.0;
        twirl_angle = twirl_angle_in * 2.0 * PI;
    }

    void evaluatePixel() ...
```

Per-frame operations - evaluateDependents

```
kernel Twirl
{
    parameter float radius_in;
    parameter float2 center_in;
    parameter float twirl_angle_in;

    dependent float radius;
    dependent float2 center;
    dependent float twirl_angle;

    void evaluateDependents()
    {
        radius = radius_in * 256.0;
        center = center_in * 512.0;
        twirl_angle = twirl_angle_in * 2.0 * PI;
    }

    void evaluatePixel() ...
```

Per-frame operations - evaluateDependents

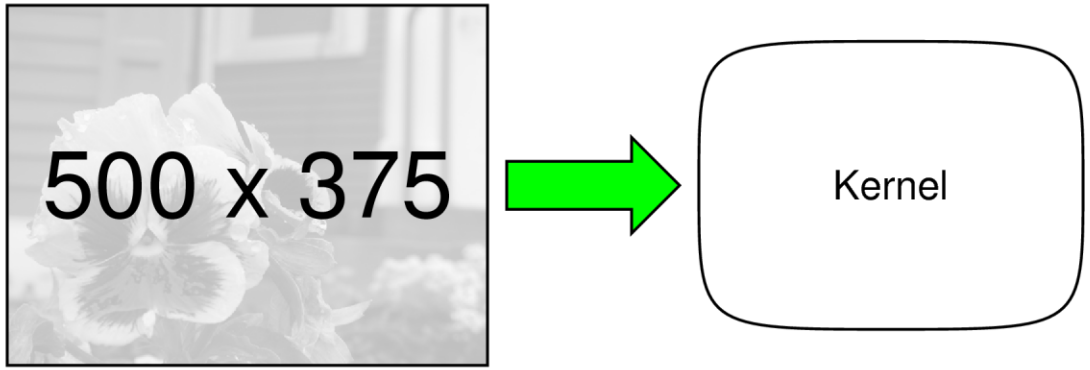
```
kernel Twirl
{
    parameter float radius_in;
    parameter float2 center_in;
    parameter float twirl_angle_in;

    dependent float radius;
    dependent float2 center;
    dependent float twirl_angle;

    void evaluateDependents()
    {
        radius = radius_in * 256.0;
        center = center_in * 512.0;
        twirl_angle = twirl_angle_in * 2.0 * PI;
    }

    void evaluatePixel() ...
```

Per-frame operations - evaluateDependents

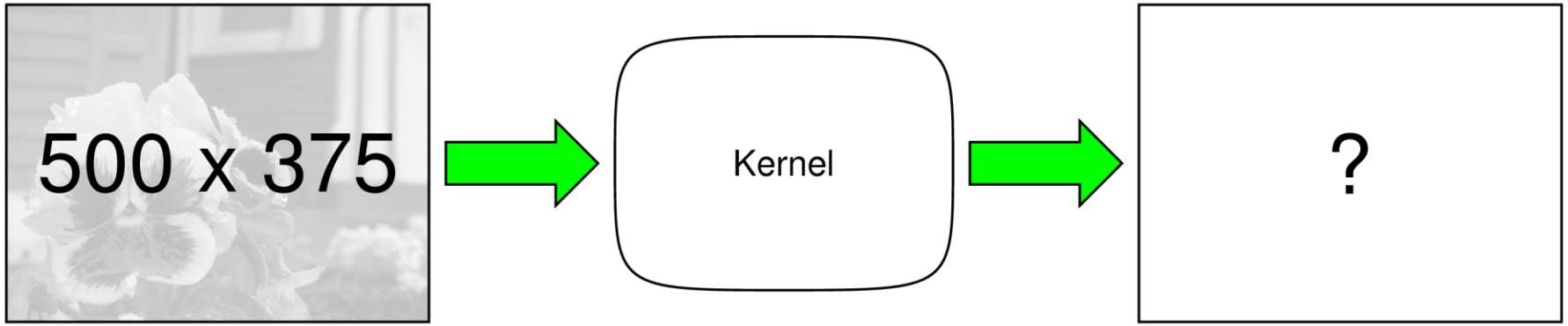
```
kernel Twirl
{
    parameter float radius_in;
    parameter float2 center_in;
    parameter float twirl_angle_in;

    dependent float radius;
    dependent float2 center;
    dependent float twirl_angle;

    void evaluateDependents()
    {
        radius = radius_in * 256.0;
        center = center_in * 512.0;
        twirl_angle = twirl_angle_in * 2.0 * PI;
    }

    void evaluatePixel() ...
```

Region Reasoning
(aka “Reason Regioning”)

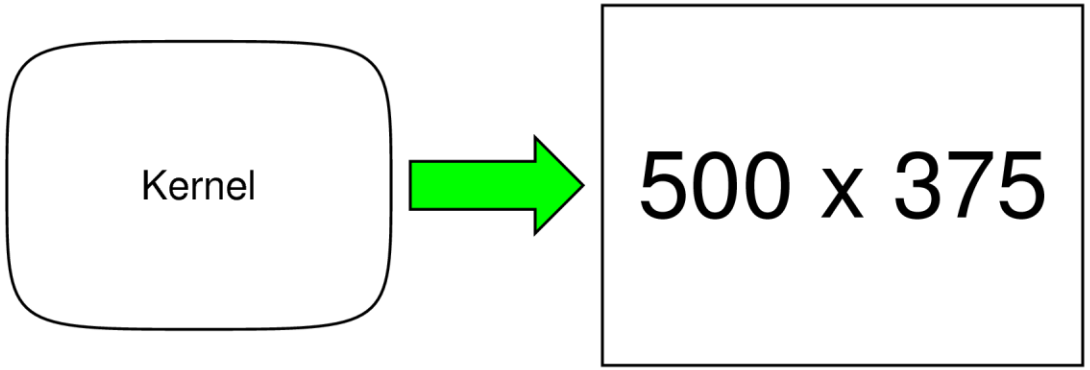
Region reasoning - changed



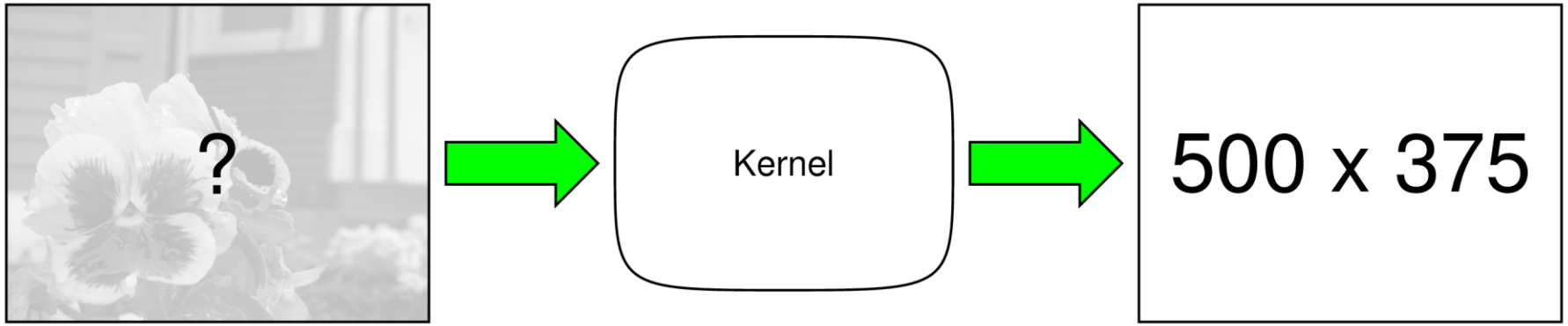
Region reasoning - changed



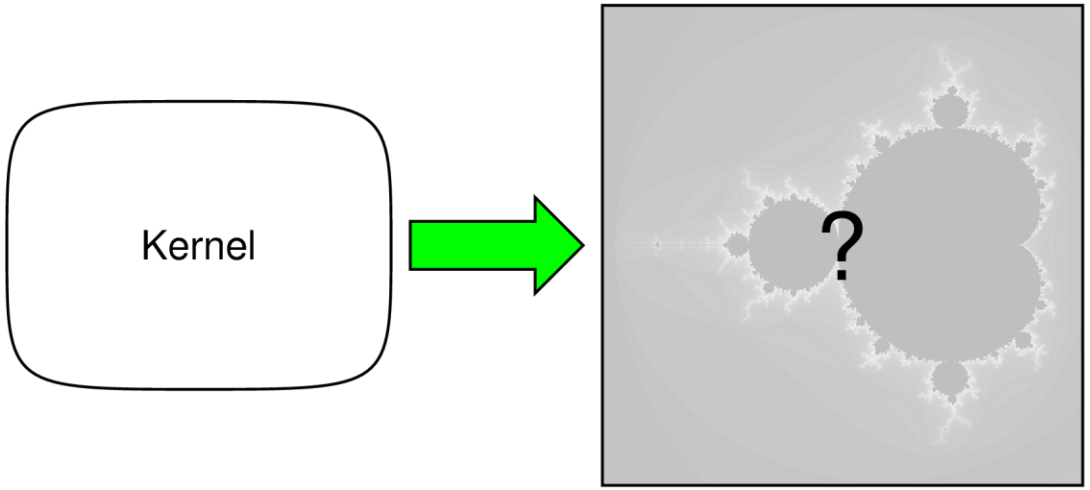
Region reasoning - needed



Region reasoning - needed



Region reasoning - generated



Region reasoning

- Changed
  - I have changed this input region, what region of the output has changed?
- Needed
  - I want this output region, what input regions do you need to have?
- Generated
  - What output region are you going to give me without any other inputs?

Region reasoning

- Domain of definition (DOD) – “what have you got?”

Region reasoning

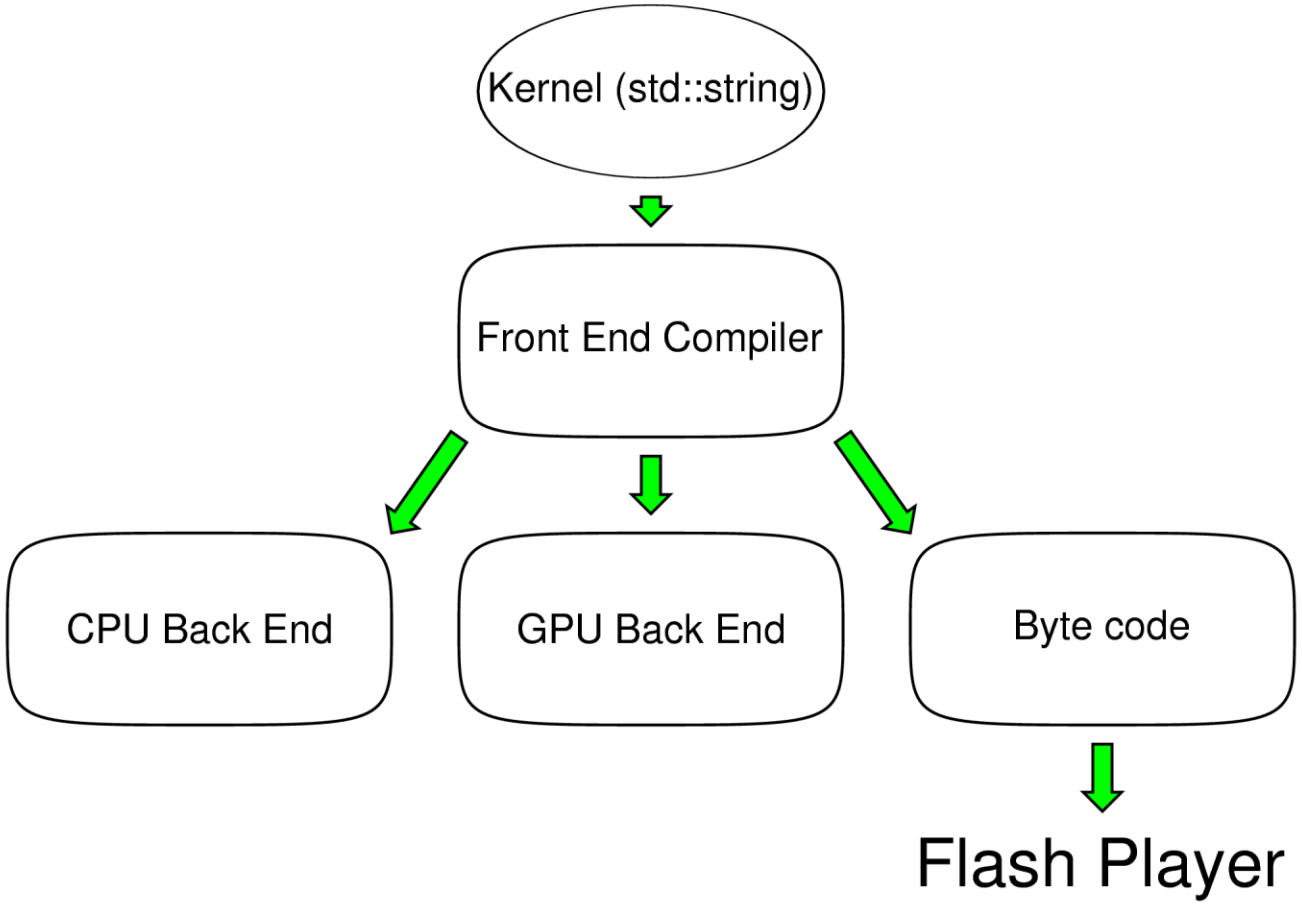
- Domain of definition (DOD) – “what have you got?”
- Region of interest (ROI) – “what do you want?”

Region reasoning

- Domain of definition (DOD) – “what have you got?”
- Region of interest (ROI) – “what do you want?”

- ROI ∩ DOD “what we’ll actually calculate”

Compiler architecture



Adobe applications using Pixel Bender

Challenges

- Wide range of hardware
  - Limits on instructions
  - Limits on instruction counts
  - Limits on support for e.g. arrays
  - Limits on image size
  - Non IEEE float
  - Criticism that the range of hardware is too wide
- Cross platform
- Cross application
- Drivers
- Read back speed
- Testing

Testing

- Programming language – infinite input space
- Automation
- Data driven
- Scripts to write tests
- Testing intermediate representations
- End to end tests
- Fuzz testing
- Test driven development
- Test for errors
- Use the documentation
- Tools

What can’t we do

- Anything that doesn’t fit our programming model
  - Histogram
  - Efficient box blur (and hence gaussian)

Where to go for information

- http://www.adobe.com/go/pixelbender_toolkit
- http://twitter.com/pixelbender
- http://blogs.adobe.com/kevin.goldsmith

- Search for
  - Pixel Bender
  - AIF
  - Adobe Image Foundation
  - Pixel Bender exchange
    - Community site for sharing Pixel Bender filters


Adobe