Applying Convolution Filters in OpenGL

Nathan Cournia

acnatha@vr.clemson.edu

Clemson University

Motivation

- Apply convolution filters:
  - On arbitrary OpenGL surfaces:
    - Primitives(lines, points, polygons)
    - Textures
    - Buffers
  - In real time

Methods

- Apply convolution via:
  - Software
  - OpenGL’s imaging subset
  - Vertex/Fragment programs

Convolution in Software

- Convolution is performed in the client software
- Gives the programmer complete control
- Image data must be downloaded/uploaded to/from HW
- Slow
- Error prone

OpenGL’s Imaging Subset

- Supported since OpenGL 1.2
- Convolution is performed through OpenGL
- Simple interface
  - `glEnable(GL_CONVOLUTION_2D)`
  - `glConvolutionFilter2D(target, iformat, w, h, format, type, kernel)`

OpenGL's Image Subset (cont.)

- Only operates on the frame buffer
- Generally not supported in HW

Convolution via Programs

- Requires OpenGL 1.5
- Uses vertex and fragment programs
- Works with all primitives
- Powerful
- Fast

Convolution: Fragment Program

- Problem:
  - Fragment programs process one fragment at a time
  - Need neighboring samples for current fragment
- Solution:
  - Pass in texture coordinates for neighboring samples

Fragment Program Texture Coords

- Problem:
  - Limited number of texture coordinate units
  - GeForceFX cards only support 8 texture coordinate units
  - We need at least 9 units for a 3x3 filter
- Solution:
  - Don’t think about texture coordinates units as storing 8 coordinates

Texture Coordinates (cont.)

- We only need 2 of the 4 components for each texture unit to sample the texture
- Treat texture coordinate units as being able to interpolate $4 * 8 = 32$ values
- We can use other interpolated variables to store more texture coordinates
  - `fragment.color.primary`
  - `fragment.color.secondary`
  - etc.

Texture Coordinate (cont.)

- We don’t need 1 texture coordinate unit for every 2 samples
- We can reuse coordinates

Texture Coordinate (cont.)

w_x, n_y	n	e_x, n_y
w	c	e
w_x, s_y	s	e_x, s_y

- For a 3x3 filter, we only need 5 texture coordinate units

Fragment Program Misc.

- Kernel weights can be passed in through local variables
- Kernel weights can also be hard-coded into the fragment program
- Separable filter simply require multiple passes

Convolution: Vertex Program

- Used to generate texture coordinate for neighboring samples

```
!!ARBvp1.0
OPTION ARB_position_invariant;
ATTRIB center = vertex.texcoord[0];
MOV result.texcoord[0], center;                  #center
ADD result.texcoord[1], center, { 1, 0, 0, 1 };  #east
ADD result.texcoord[2], center, { -1, 0, 0, 1 }; #west
ADD result.texcoord[3], center, { 0, 1, 0, 1 };  #north
ADD result.texcoord[4], center, { 0, -1, 0, 1 }; #south
END
```

Convolution: Vertex Program (cont.)

- Not really necessary (can be done in client program)
- Notice vertex program add 1/-1 to texture coordinate components
  - Facilitated via `GL_EXT_texture_rectangle` texture target

GL_EXT_texture_rectangle

- Allows access to texture via coordinates in the range $[0,w]$, $[0,h]$ where w,h are the respective width and height of the image
- Appears to be the same extension as `GL_NV_texture_rectangle`
- Limited wrap states
- Mipmaps can not be defined

Fragment Program Example

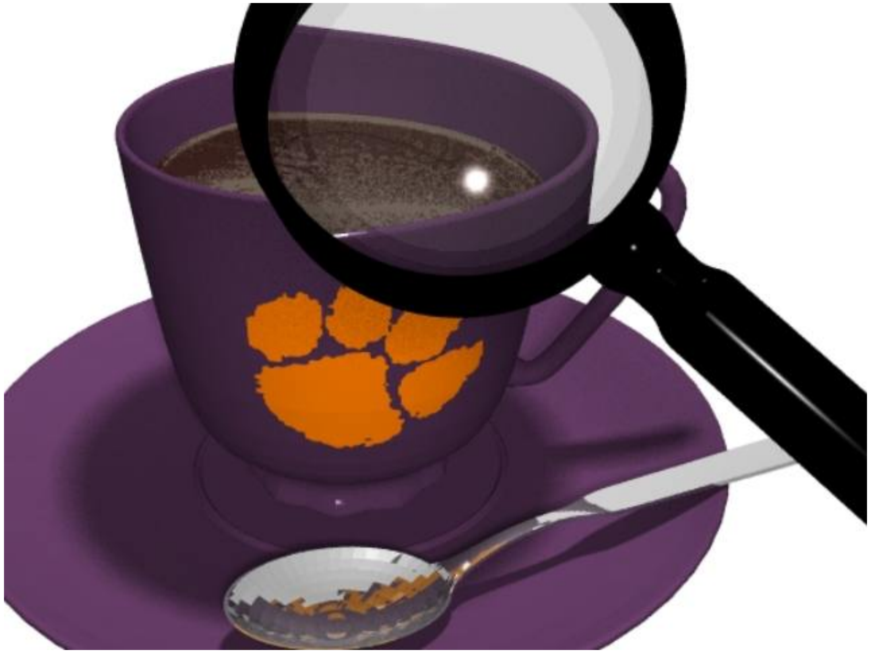
```
!!ARBfp1.0
ATTRIB center = fragment.texcoord[0];
ATTRIB east = fragment.texcoord[1];   #... west, north, south

PARAM w0 = program.local[0];          #kernel (nw,n,ne,w)
PARAM w1 = program.local[1];          #       (c, e,sw,s)
PARAM w2 = program.local[2];          #       (se,scale,bias,0)

TEX sample, center, texture[0], RECT; #center
MUL sum, sample, w1.x;
TEX sample, north, texture[0], RECT;  #north
MAD sum, sample, w0.y, sum;           #...

MOV_SAT result.color, sum;           #output final color
END
```

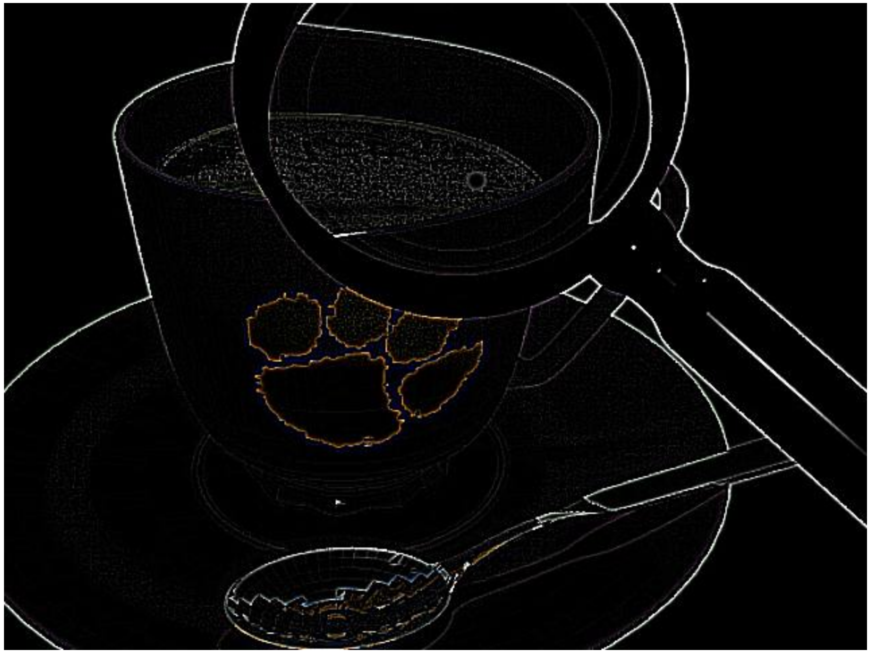
Edge Detection

-1	-1	-1
-1	8	-1
-1	-1	-1

- 3x3 Laplacian Kernel

Edge Detection Results



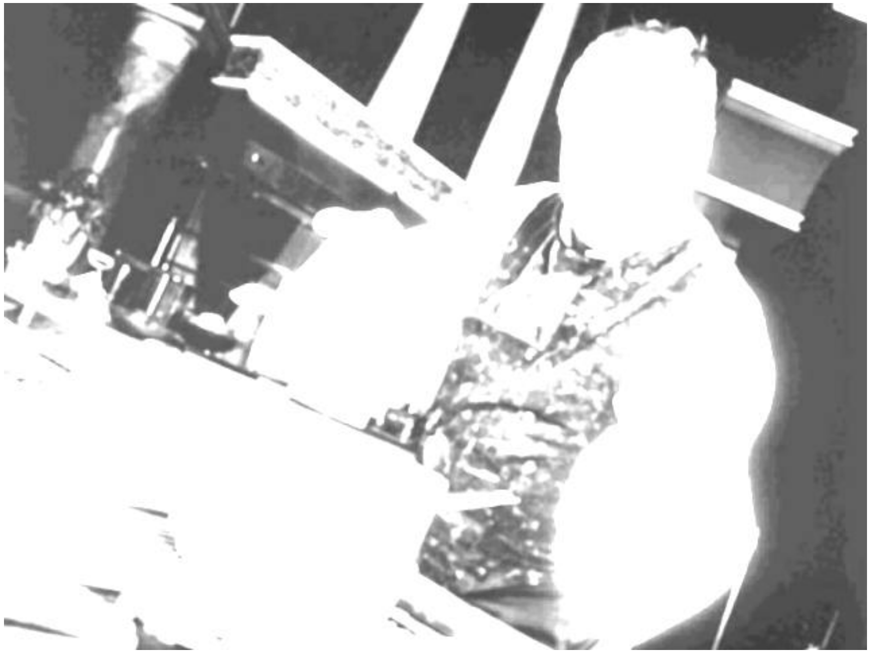


Smoothing

1	2	1
2	4	2
1	2	1

- 3x3 Smoothing Kernel

Smoothing Results



Conclusion

- Convolution in OpenGL is possible at interactive frame rates
- OpenGL's imaging subset convolution is nice, but requires expensive hardware
- Fragment programs make convolution highly configurable and easy to use

Questions

- Is it possible to do convolution with 3D images? Does it make sense?
- What will GLslang bring to the table?