GPU-Accelerated Iterated Function Systems

Simon Green, NVIDIA Corporation
Iterated Function Systems

- Fractal
- Conceived by John Hutchinson (1981)
  - Popularized by Michael Barnsley (*Fractals Everywhere*, 1998)
- Consists of a set of functions
  - Functions map points from one point in space to another
  - Traditionally functions are affine transformations
  - Should be contractive (move points closer together)
• Image is set of points that are the solution to recursive set equation:

\[
S = \bigcup_{i=0}^{n-1} f_i(S) \quad \text{where} \quad S \in \mathbb{R}^2 \quad f_i: \mathbb{R}^2 \rightarrow \mathbb{R}^2
\]

\[
f_i(x,y) = (a_i x + b_i y + c_i, d_i x + e_i y + f_i)
\]
Generating Images of an IFS

• Commonly solved using the “chaos game”

  Pick random point in space
  Repeat {
    Pick function at random
    Apply function to point
    Plot the point
  }

• Converges to correct solution in ~20 steps
Example: Sierpinski Triangle

\[ F_0(x, y) = \left( \frac{x}{2}, \frac{y}{2} \right) \]
\[ F_1(x, y) = \left( \frac{x+1}{2}, \frac{y}{2} \right) \]
\[ F_2(x, y) = \left( \frac{x}{2}, \frac{y+1}{2} \right) \]
Example: Sierpinski Triangle

\[
\begin{align*}
F_0(x, y) &= \left(\frac{x}{2}, \frac{y}{2}\right) \\
F_1(x, y) &= \left(\frac{x+1}{2}, \frac{y}{2}\right) \\
F_2(x, y) &= \left(\frac{x}{2}, \frac{y+1}{2}\right)
\end{align*}
\]
Example: Sierpinski Triangle

\[ F_0(x, y) = \left( \frac{x}{2}, \frac{y}{2} \right) \]
\[ F_1(x, y) = \left( \frac{x+1}{2}, \frac{y}{2} \right) \]
\[ F_2(x, y) = \left( \frac{x}{2}, \frac{y+1}{2} \right) \]
Example: Sierpinski Triangle

\[ F_0(x, y) = \left( \frac{x}{2}, \frac{y}{2} \right) \]
\[ F_1(x, y) = \left( \frac{x+1}{2}, \frac{y}{2} \right) \]
\[ F_2(x, y) = \left( \frac{x}{2}, \frac{y+1}{2} \right) \]
Example: Sierpinski Triangle

\[ F_0(x, y) = \left( \frac{x}{2}, \frac{y}{2} \right) \]
\[ F_1(x, y) = \left( \frac{x+1}{2}, \frac{y}{2} \right) \]
\[ F_2(x, y) = \left( \frac{x}{2}, \frac{y+1}{2} \right) \]
Fractal Flames

• Invented by Scott Draves
• Used in “Electric Sheep” screen saver
• Modification of traditional IFS:
  – Non-linear functions as well as affine transforms
  – Log-density display
  – Color by structure
  – Animates transformations and function weights
Fractal Flames
GPU Particle Systems

- Render-to-vertex array made particle systems possible on the GPU
  - Lutz Latta, “Building a Million Particle System”, GDC 2004
- Positions / velocities of particles stored in textures
- Simulation of particles computed using fragment programs
- Results rendered directly as point-sprites
  - No read-back to CPU required
- Enables millions of particles at interactive speeds
GPUflame

- Uses the GPU to accelerate computation and display of IFS fractals
- Extends functions to 3 dimensions
- Uses floating point blending for HDR display
GPUflame Computation

- Uses standard GPGPU techniques
- Point positions stored in floating point texture
- Single quad drawn per iteration
- Fragment program applies IFS functions
  - Uses Cg language to describe functions
  - Random numbers come from pre-calculated texture
GPUflame Colouring

• Uses color index value
  – Tracks number of function applied at each step
  – Stored in alpha channel of position texture

• Value is used in final point shader to index into 1D palette texture
Colour Palettes
Colour Palettes
Colour Palettes
// interface for variations  
interface Variation {
    float3 func(float3 p, float r, float theta);
};

float3 linear(float3 p, float r, float theta)
{
    return p;
}

float3 spherical(float3 p, float r, float theta)
{
    return normalize(p);
}

float3 swirl(float3 p, float r, float theta)
{
    return float3(r*cos(theta+r), r*sin(theta+r), p.z);
}

...
#include "variations.cg"

float4 main(in float2 uv : TEXCOORD0,
            uniform samplerRECT pos_tex,
            uniform samplerRECT rand_tex,
            uniform float time,
            uniform float4x4 matrix[],
            uniform Variation variation
        ) : COLOR
{
  // get position and color
  float4 tex = texRECT(pos_tex, uv);
  float3 p = tex.xyz;
  float c = tex.w;

  // get random number
  float4 rand = tex2D(rand_tex, (uv + float2(time, 0)));
// apply transformation
int ri = floor(rand.x * matrix.length);
for(int i=0; i<matrix.length; i++) {
    if (ri==i) {
        p = mul(matrix[i], float4(p, 1.0));
        ci = i / (float) (matrix.length-1);
    }
}

// apply function
float r = length(p);
float theta = atan2(p.x, p.y);
p = variation.func(p, r, theta);

c = (c + ci) / 2.0;    // mix old and new color
return float4(p, c);
GPUflame Rendering

- Points rendered using render-to-vertex array (VBO/PBO OpenGL extensions)
  - Use of point sprites possible
- Rendered to 16-bit floating point buffer
  - Using additive blending
Tone Mapping

• Simple tone mapping pass
  – Converts floating point values to displayable 8-bit values
  – Exposure and gamma controls

• Glow effect
  – Perform Gaussian blur on image
  – Mix original and blurred image
  – Helps communicate very bright regions
Without Tone Mapping
With Tone Mapping
Without Glow
With Glow
Performance

• 1 million points @ 45 frames/sec
  – GeForce 6800 Go
  – 3 transforms, 1 function

• High quality rendering @ 10 frames/sec
  – 256k points, 10 passes

• Every function must be evaluated for each point
Future Work

• Analytical point anti-aliasing
• Tiled rendering for high resolution prints
• More complex colouring schemes
• New tone mapping algorithms
• New IFS functions
Conclusion

• GPU hardware allows interactive exploration of a complex mathematical space

• Questions?