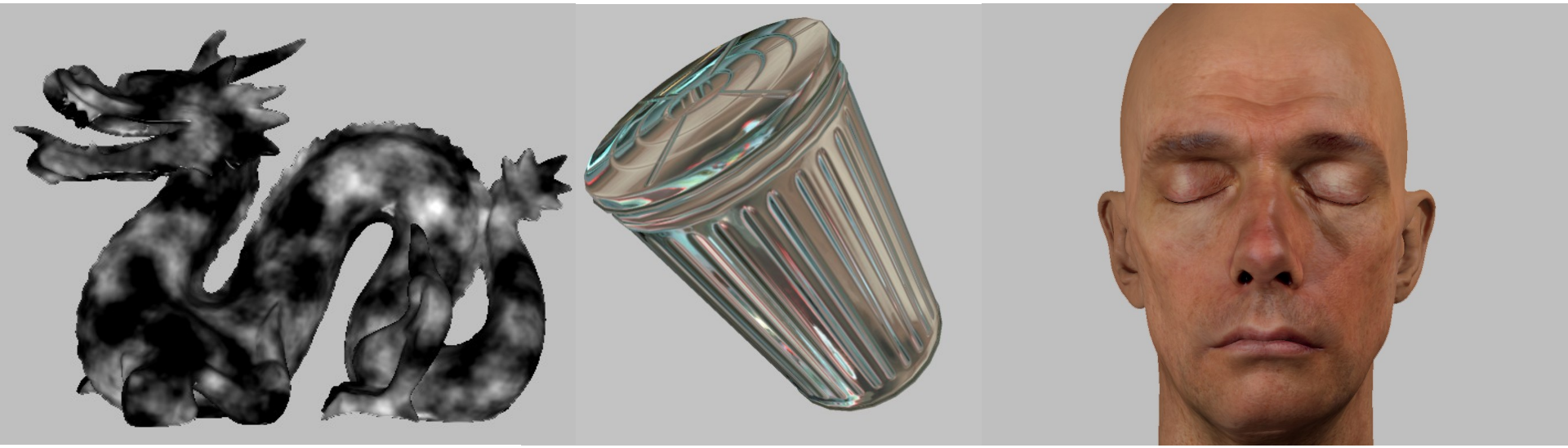
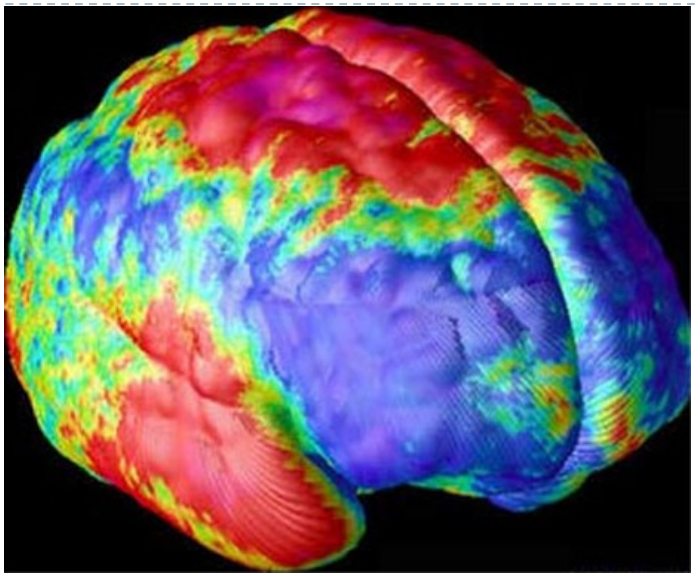


Genetic Programming for Shader Simplification

(UVA TR CS-2011-03)

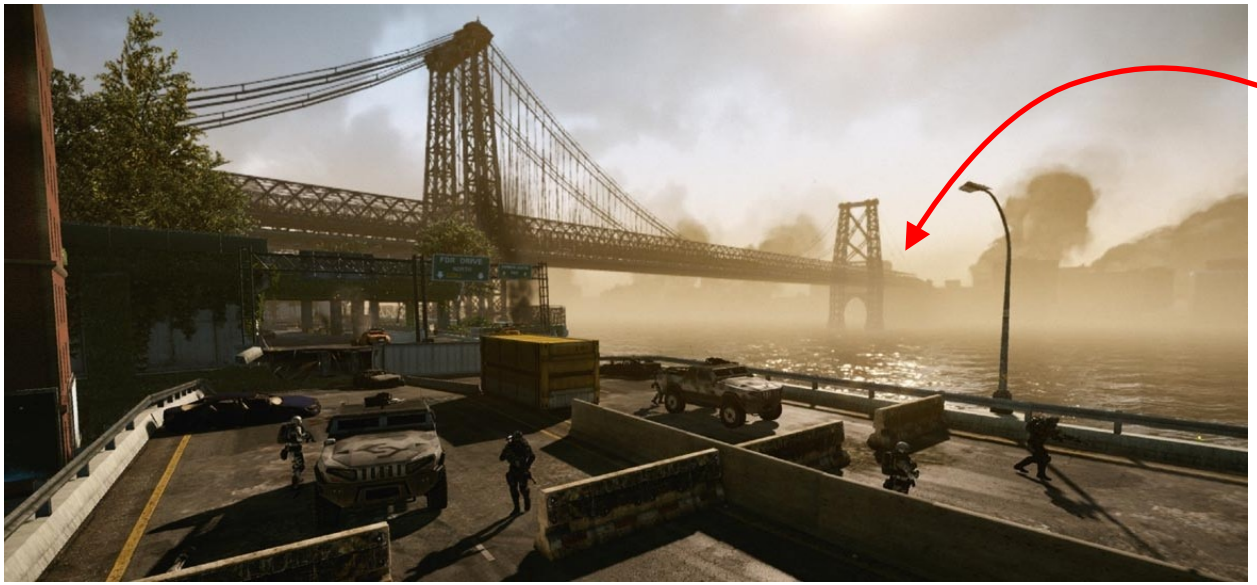
Pitchaya Sitthi-amorn,
Nick Modly, Jason Lawrence,
Westley Weimer

Motivation: Real-Time Rendering



Motivation: Pixel Shaders

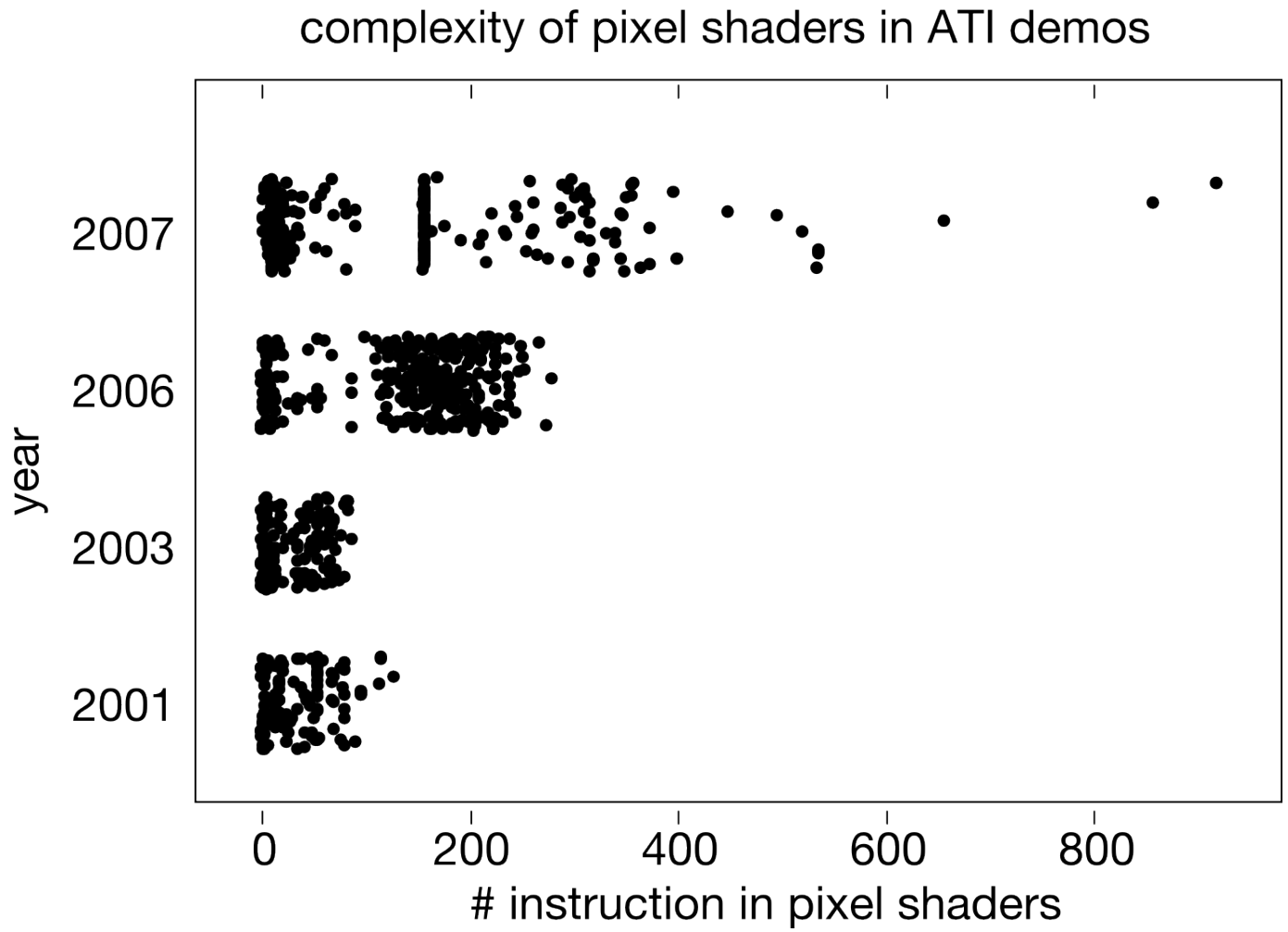
- Most effects and computations occur at the pixel level
- Pixel shader: a user program that executes at each pixel



- Fog
- Lighting
- Water ripples
- Soft shadows
- Environment reflection mapping

[Crysis 2]

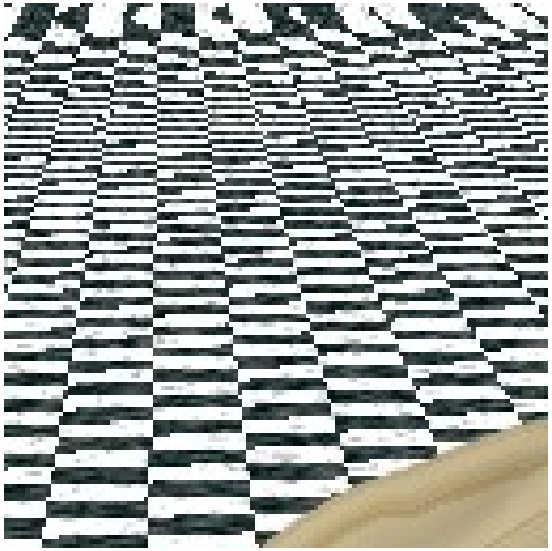
Motivation: Shader Complexity



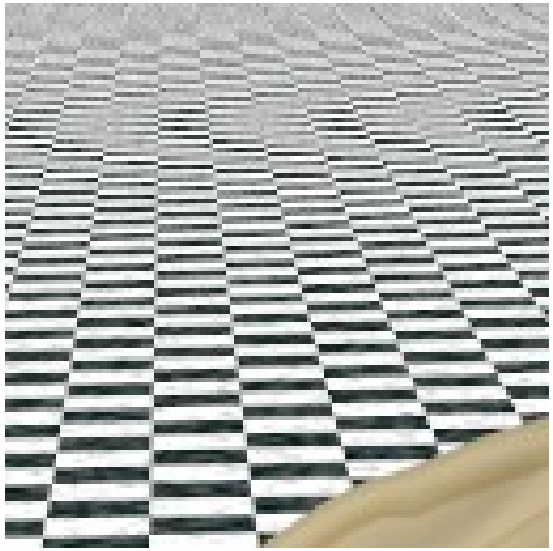
[Norman Rubin]

Motivation: Shader Run Time

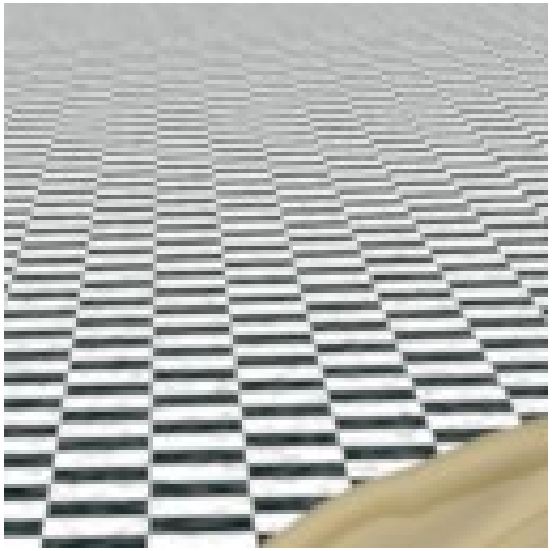
- Pixel shaders are executed several times to avoid aliasing



1 sample



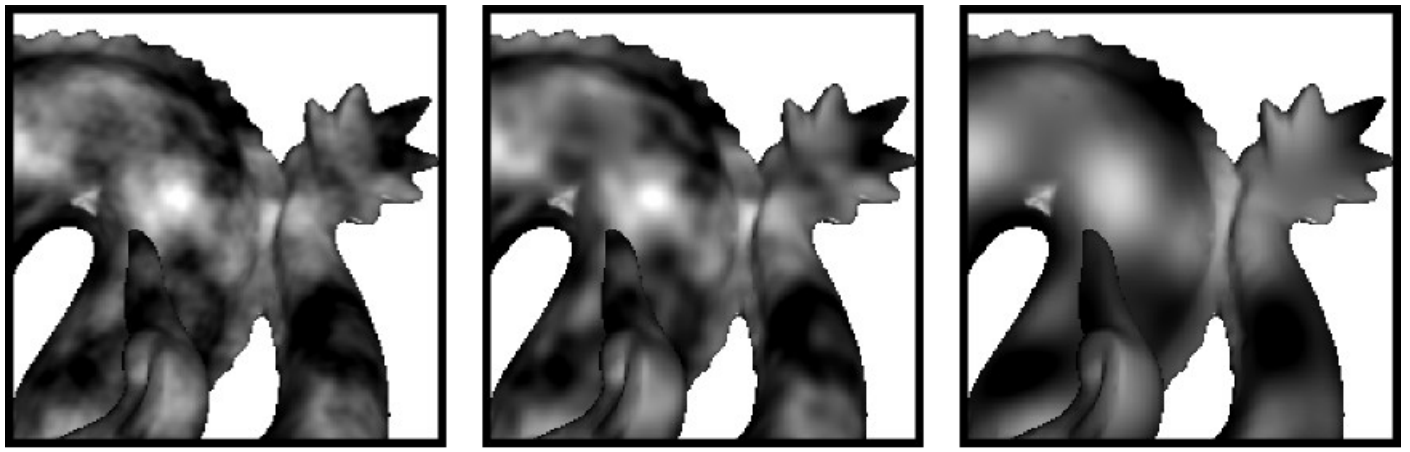
16 samples



256 samples

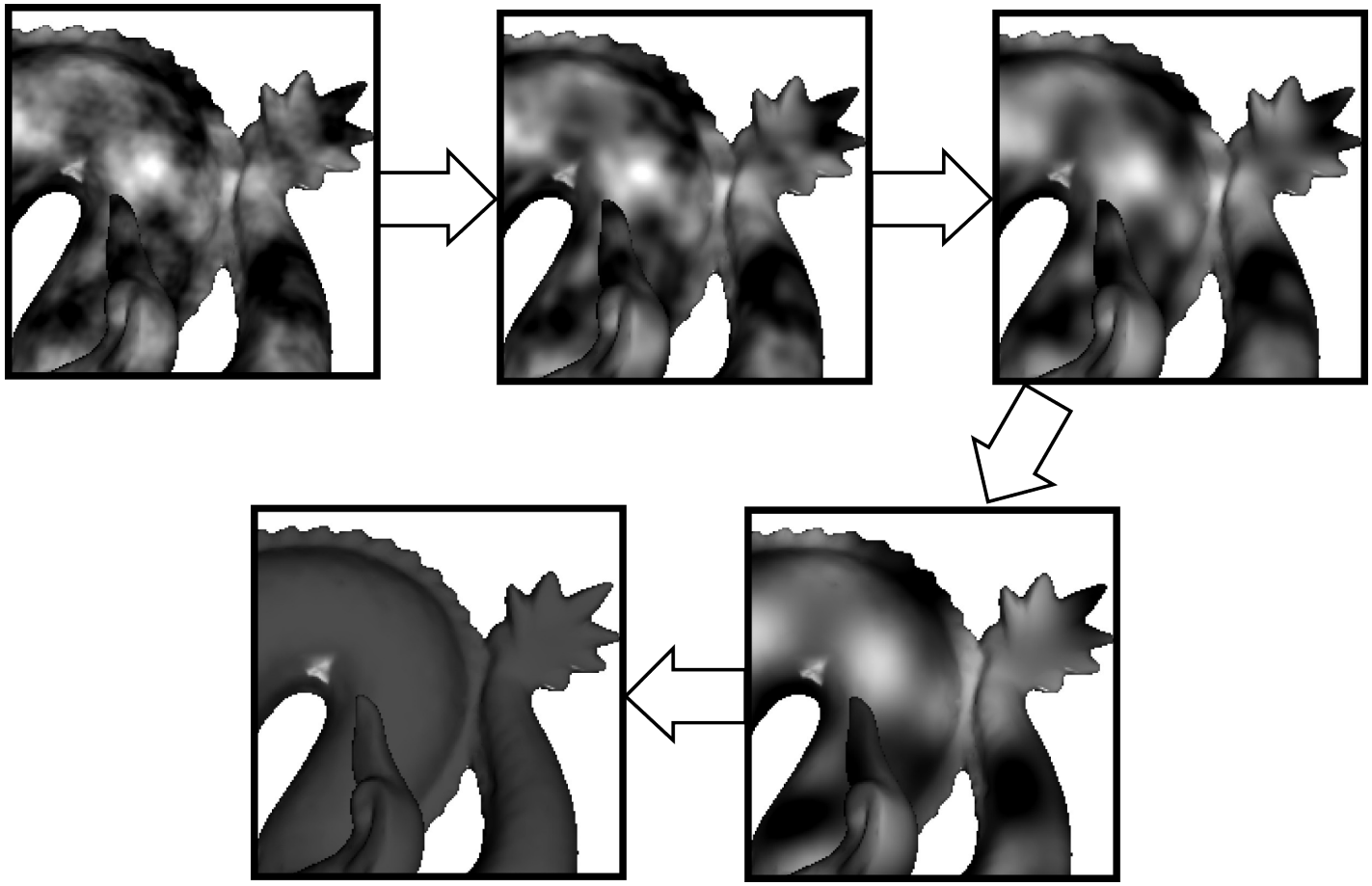
Key Insight

- Most shaders can be simplified with an acceptable loss in detail
- Output is consumed by human eyes



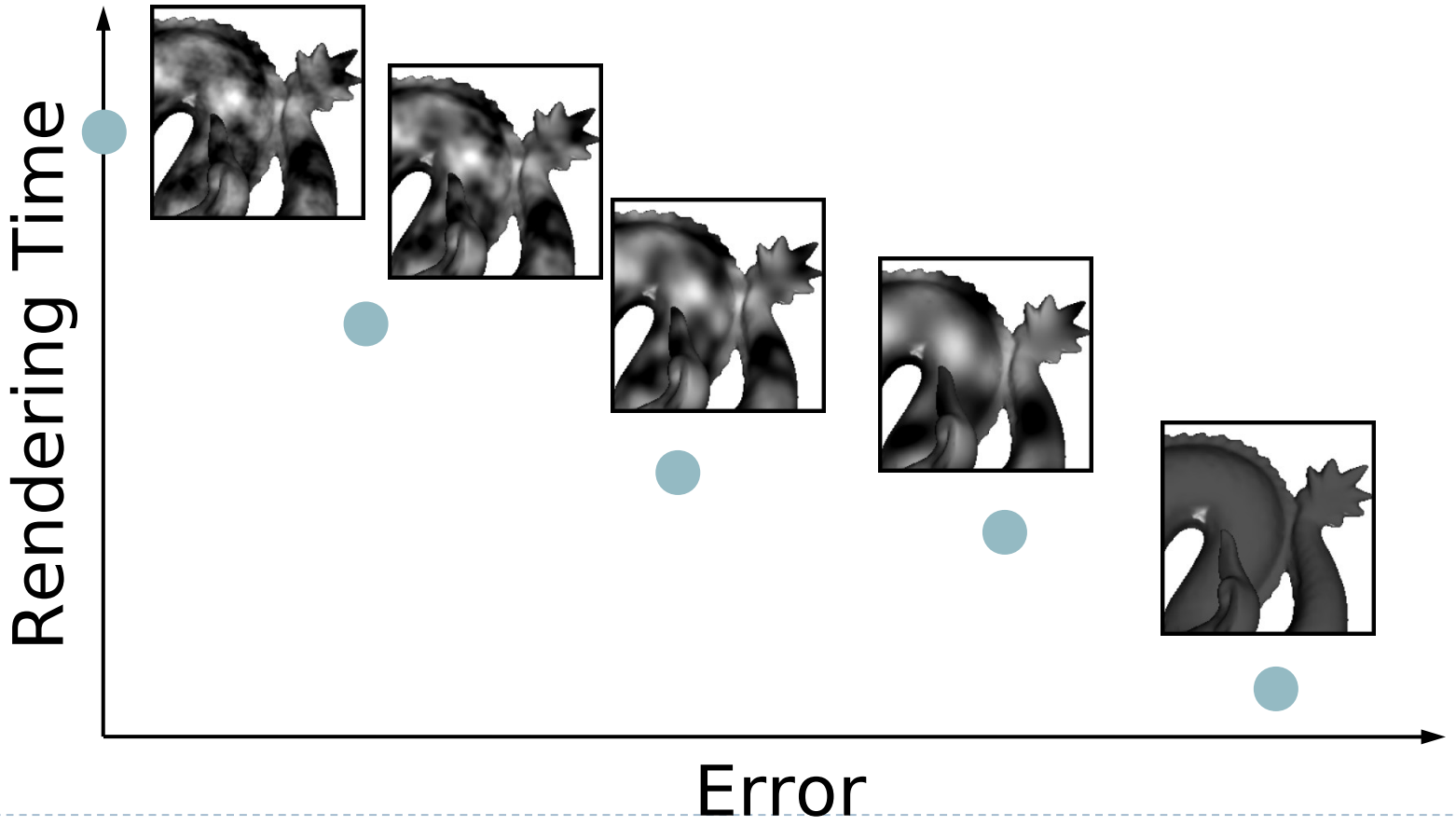
Problem Statement

- Generate a sequence of simplified shaders



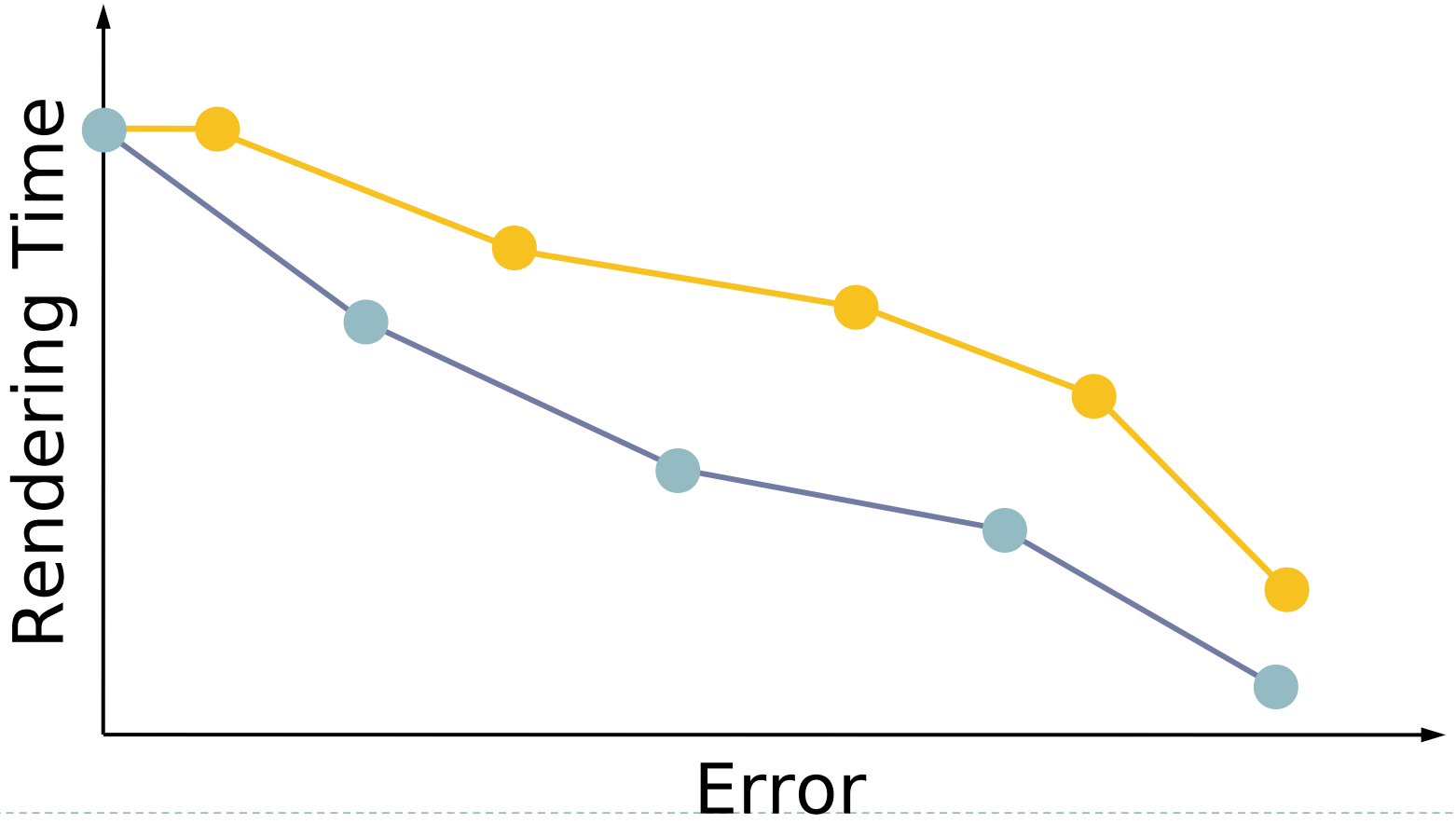
Problem Statement

- Generate sequence of simplified shaders



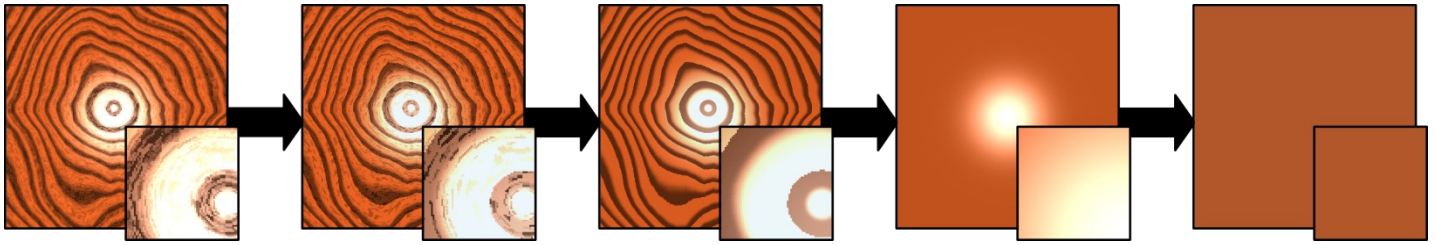
Problem Statement

- Generate sequence of simplified shaders



Previous Work

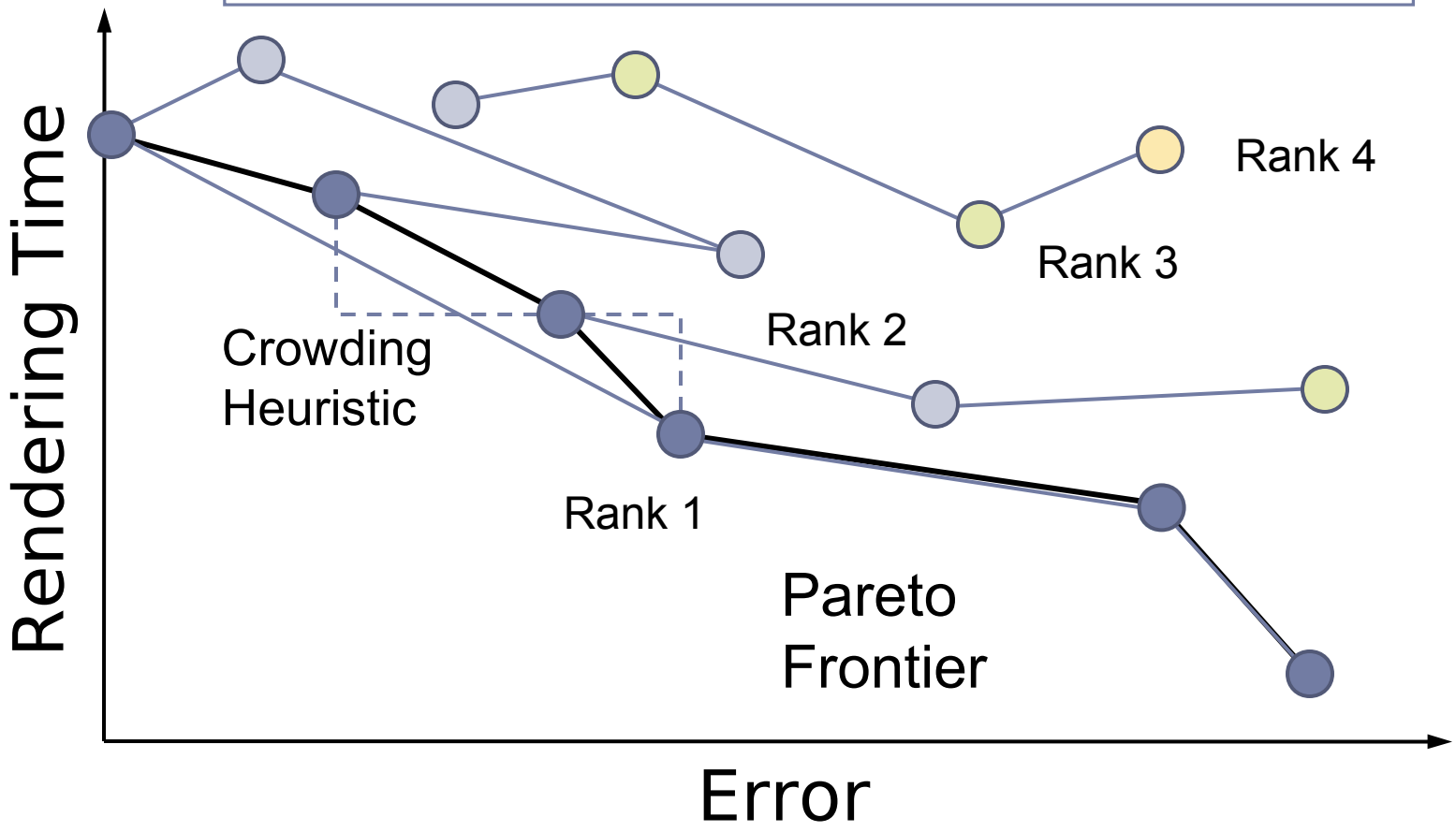
Olano et. al [2003]



Pellacini [2005]

Multi-Objective Genetic Programming (NSGA-II):

Initialize Sort Select Pairs Crossover Mutate-Evaluate Select



Mutation Operator: Insert

```
1.float Fresnel (float th, float n) {
2.    float cosi = cos (th);
3.    float R = 1.0f;
4.    float n12 = 1.0f / n;
5.    float sint = n12 * sqrt (1 - (cosi * cosi));
6.    if (sint < 1.0f) {
7.       float cost = sqrt (1.0 - (sint * sint));
8.       float r_ortho = (cosi - n * cost)
9.          / (cosi + n * cost);
10.       float r_par = (cost - n * cosi)
11.          / (cost + n * cosi);
12.       R=(r_ortho * r_ortho + r_par * r_par)/2;
13.    }
14.    return R;
15.}
```

Mutation Operator: Insert

```
1.float Fresnel (float th, float n) {
2.    float cosi = cos (th);
3.    float R = 1.0f;
4.    float n12 = 1.0f / n;
5.    float sint = n12 * sqrt (1 - (cosi * cosi));
6.    if (sint < 1.0f) {
7.       float cost = sqrt (1.0 - (sint * sint));
8.       float r_ortho = (cosi - n * cost)
9.          / (cosi + n * cost);
10.       float r_par = (cost - n * cosi)
11.          / (cost + n * cosi) - (cosi * cosi);
12.       R=(r_ortho * r_ortho + r_par * r_par)/2;
13.    }
14.    return R;
15.}
```

Mutation Operator: Delete

```
1.float Fresnel (float th, float n) {
2.    float cosi = cos (th);
3.    float R = 1.0f;
4.    float n12 = 1.0f / n;
5.    float sint = n12 * sqrt (1 - (cosi * cosi));
6.    if (sint < 1.0f) {
7.       float cost = sqrt (1.0 - (sint * sint));
8.       float r_ortho = (cosi - n * cost)
9.          / (cosi + n * cost);
10.       float r_par = (cost - n * cosi)
11.          / (cost + n * cosi);
12.       R=(r_ortho * r_ortho + r_par * r_par)/2;
13.    }
14.    return R;
15.}
```

Mutation Operator: Delete

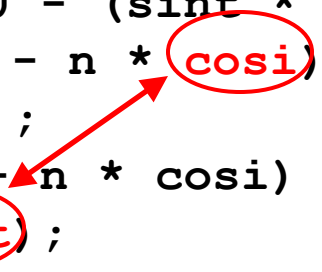
```
1.float Fresnel (float th, float n) {
2.    float cosi = cos (th);
3.    float R = 1.0f;
4.    float n12 = 1.0f / n;
5.    float sint = n12 * sqrt (1 - (cosi * cosi));
6.    if (sint < 1.0f) {
7.       float cost = sqrt (1.0 - (sint * sint));
8.       float r_ortho = (cosi - n * cost)
9.          / (cosi + n * cost);
10.       float r_par = (cost - n * cosi)
11.          / (cost + n * cosi);
12.       R=(r_ortho * r_ortho + r_par * r_par)/2;
13.    }
14.    return R;
15.}
```

Mutation Operator: Swap

```
1.float Fresnel (float th, float n) {
2.    float cosi = cos (th);
3.    float R = 1.0f;
4.    float n12 = 1.0f / n;
5.    float sint = n12 * sqrt (1 - (cosi * cosi));
6.    if (sint < 1.0f) {
7.       float cost = sqrt (1.0 - (sint * sint));
8.       float r_ortho = (cosi - n * cost)
9.          / (cosi + n * cost);
10.       float r_par = (cost - n * cosi)
11.          / (cost + n * cosi);
12.       R=(r_ortho * r_ortho + r_par * r_par)/2;
13.    }
14.    return R;
15.}
```

Mutation Operator: Swap

```
1.float Fresnel (float th, float n) {
2.    float cosi = cos (th);
3.    float R = 1.0f;
4.    float n12 = 1.0f / n;
5.    float sint = n12 * sqrt (1 - (cosi * cosi));
6.    if (sint < 1.0f) {
7.       float cost = sqrt (1.0 - (sint * sint));
8.       float r_ortho = (cosi - n * cosi)
9.          / (cosi + n * cost);
10.       float r_par = (cost - n * cosi)
11.          / (cost + n * cost);
12.       R=(r_ortho * r_ortho + r_par * r_par)/2;
13.    }
14.    return R;
15.}
```



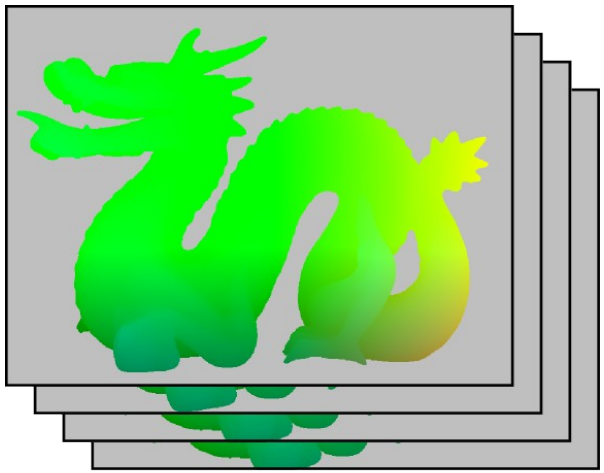
Mutation Operator: Replacing with its average value

```
1.float Fresnel (float th, float n) {
2.    float cosi = cos (th);
3.    float R = 1.0f;
4.    float n12 = 1.0f / n;
5.    float sint = n12 * sqrt (1 - (cosi * cosi));
6.    if (sint < 1.0f) {
7.       float cost = sqrt (1.0 - (sint * sint));
8.       float r_ortho = (cosi - n * cost)
9.          / (cosi + n * cost);
10.       float r_par = (cost - n * cosi)
11.          / (cost + n * cosi);
12.       R=(r_ortho * r_ortho + r_par * r_par)/2;
13.    }
14.    return R;
15.}
```

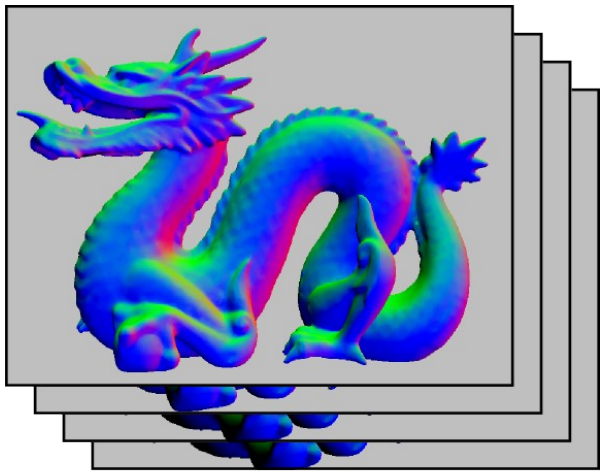
Mutation Operator: Replacing with the average value

```
1.float Fresnel (float th, float n) {
2.    float cosi = cos (th);
3.    float R = 1.0f;
4.    float n12 = 1.0f / n;
5.    float sint = n12 * sqrt (1 - (cosi * cosi));
6.    if (sint < 1.0f) {
7.       float cost = sqrt (1.0 - (sint * sint));
8.       float r_ortho = (cosi - n * cost)
9.          / (cosi + n * cost);
10.       float r_par = (cost - n * cosi)
11.          / (cost + n * 0.5);
12.       R=(r_ortho * r_ortho + r_par * r_par)/2;
13.    }
14.    return R;
15.}
```

Measuring Error/Performance

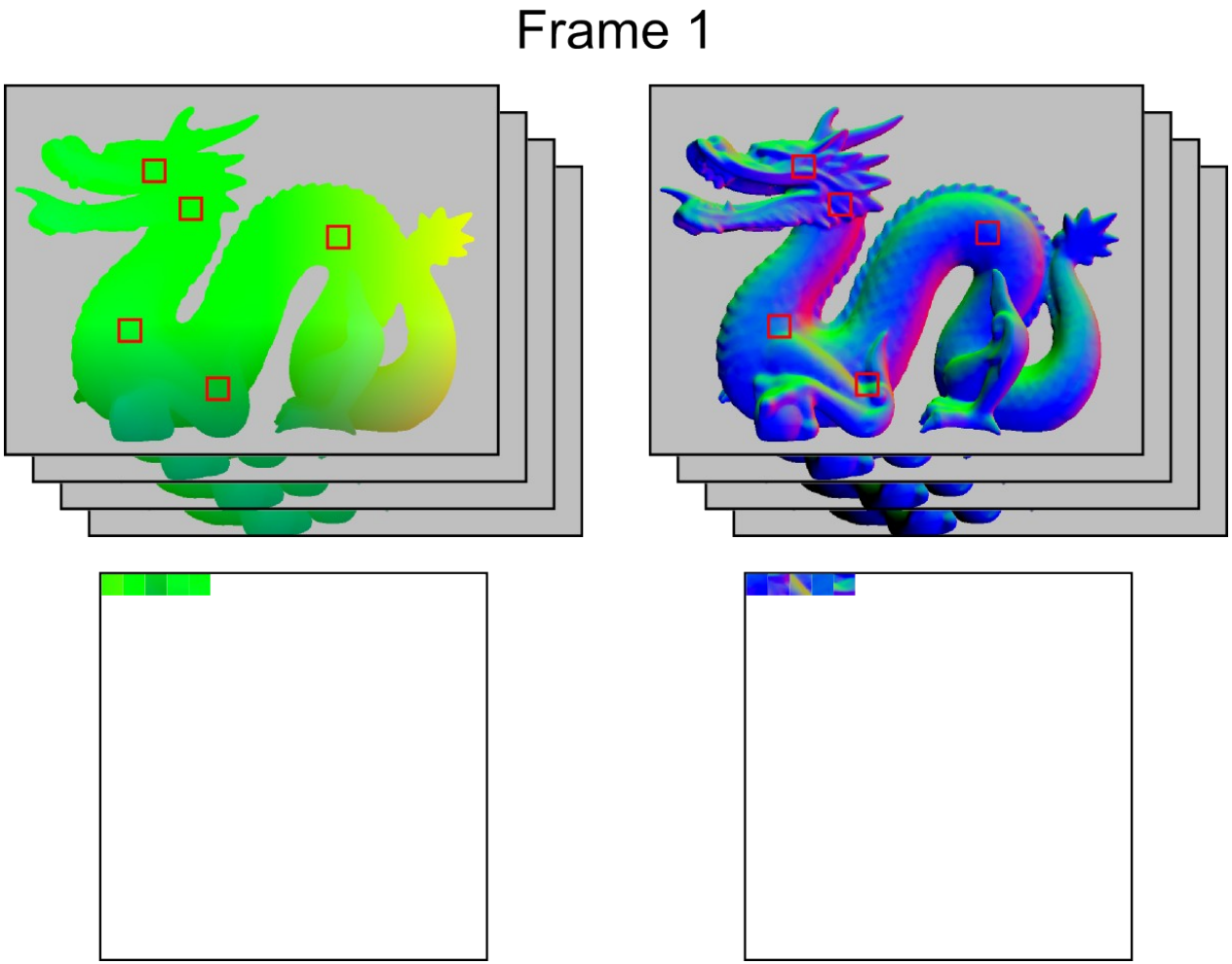


Original

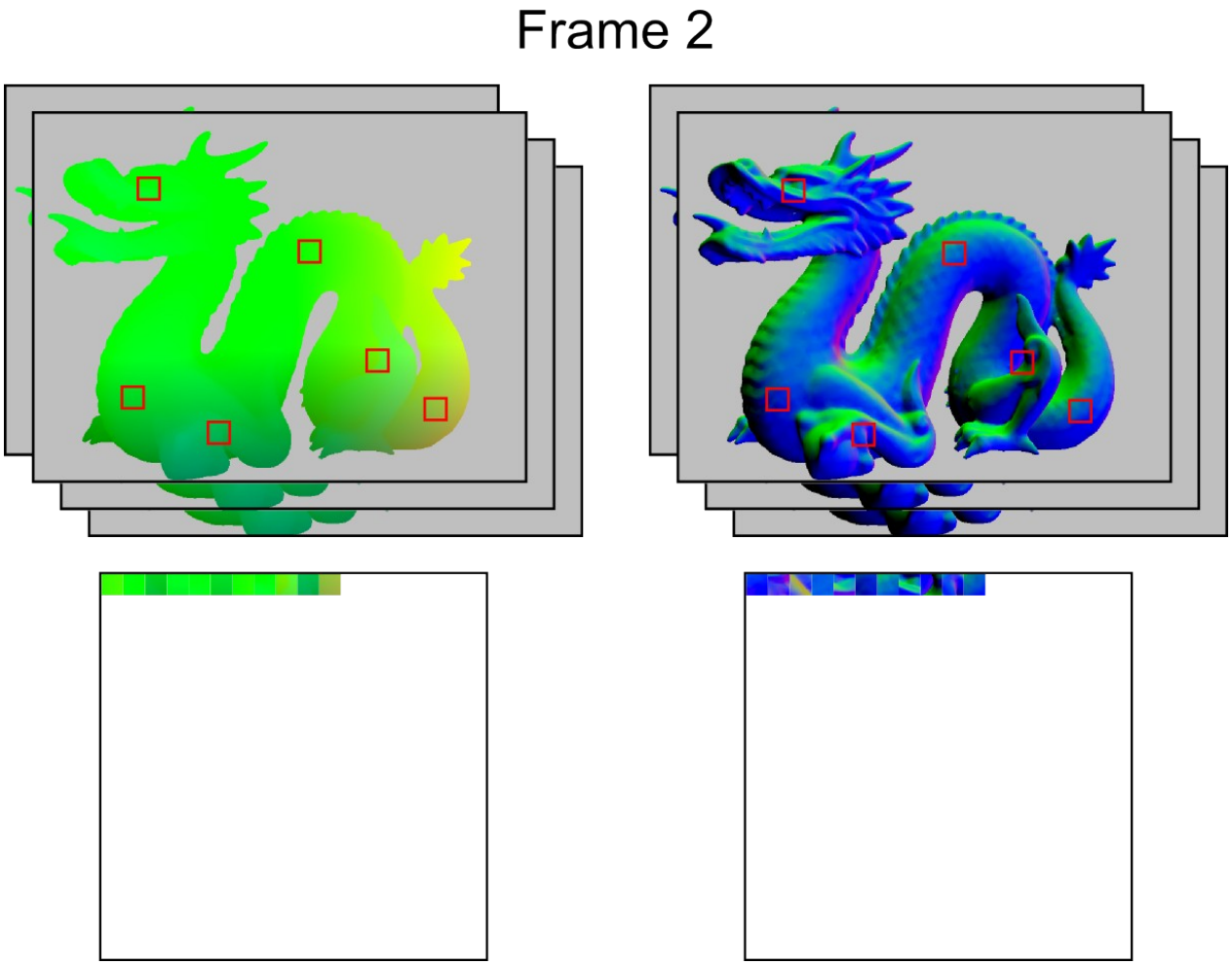


Modified

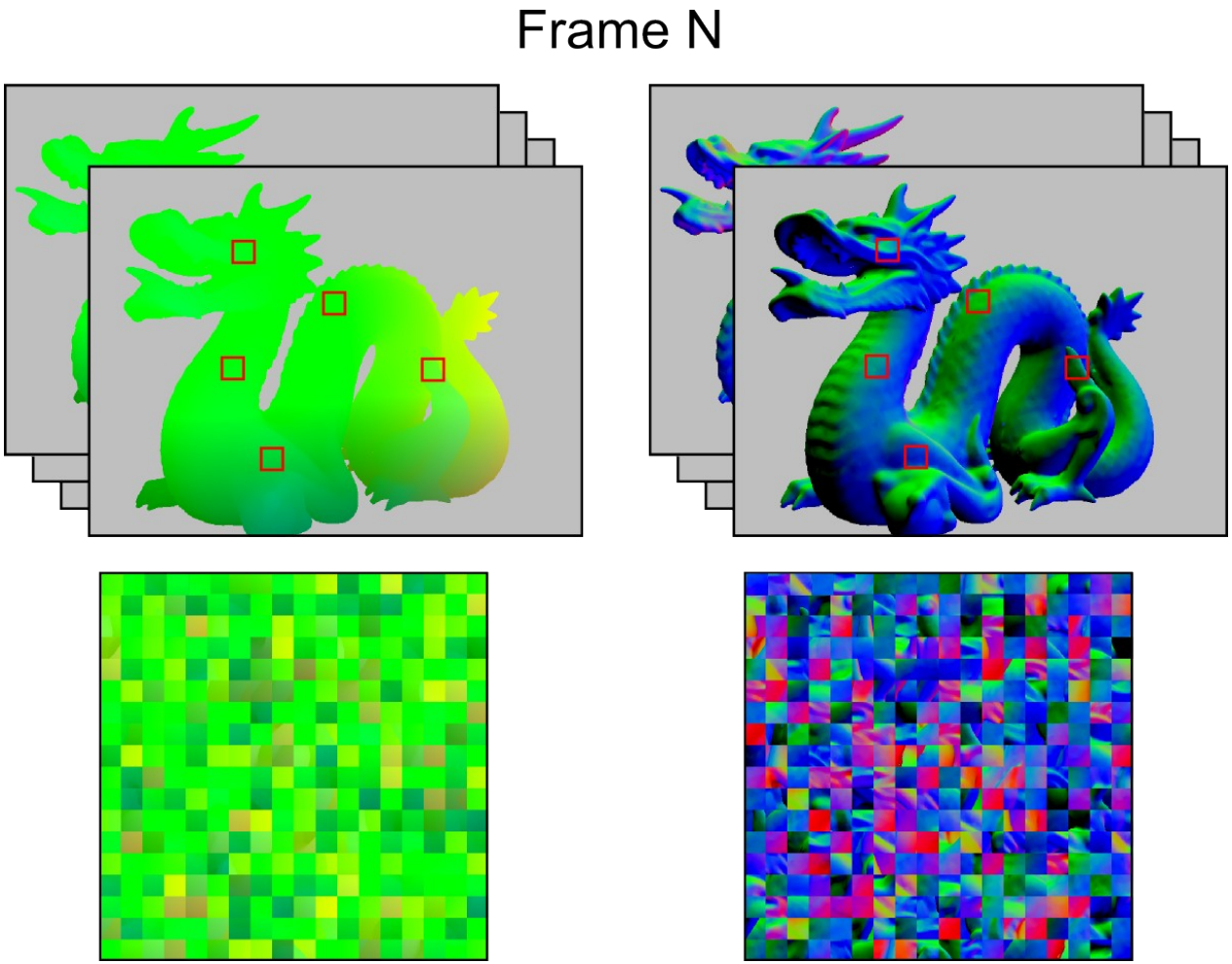
Approximating Error/Performance



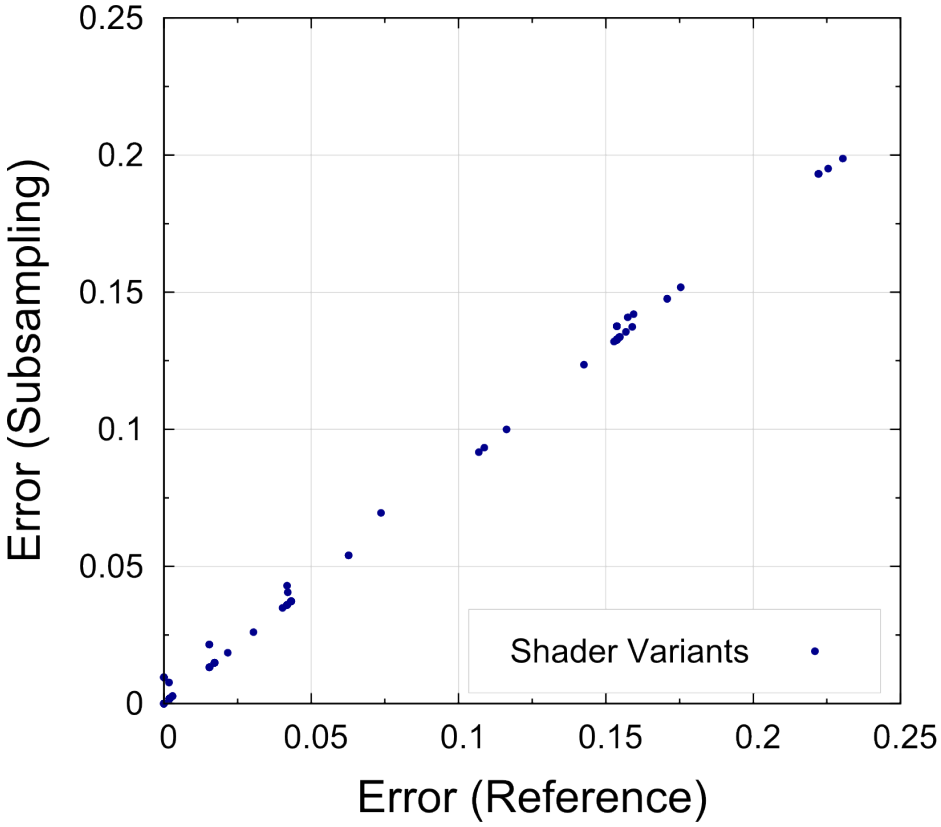
Approximating Error/Performance

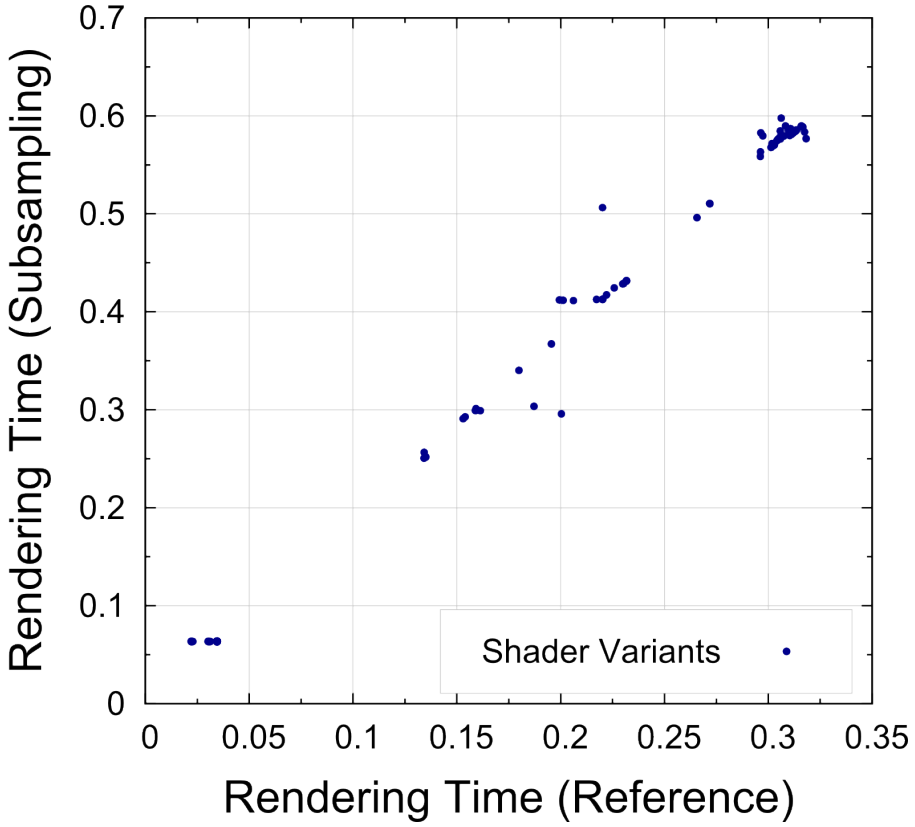


Approximating Error/Performance



Error/Performance Approximation Validation

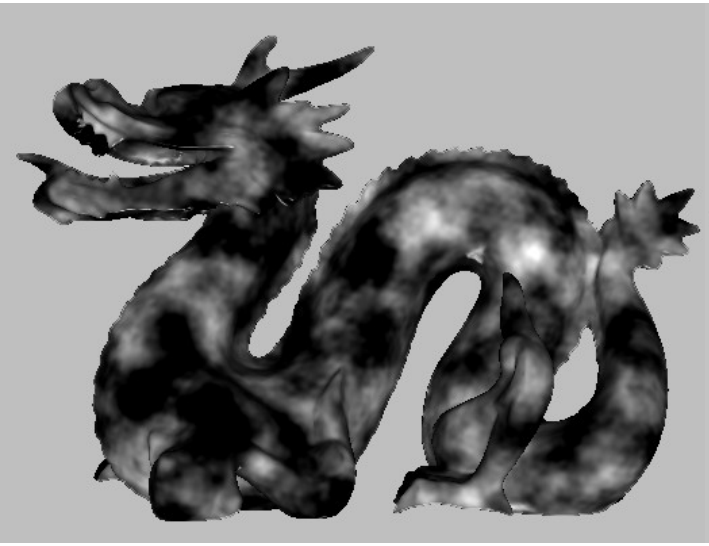




Cross correlation 0.84 and 0.95
Fitness evaluation time improvement: 100x

Test Scenes and Shaders

Marble shader



procedural noise with
Blinn-Phong specular layer
(75K triangles)

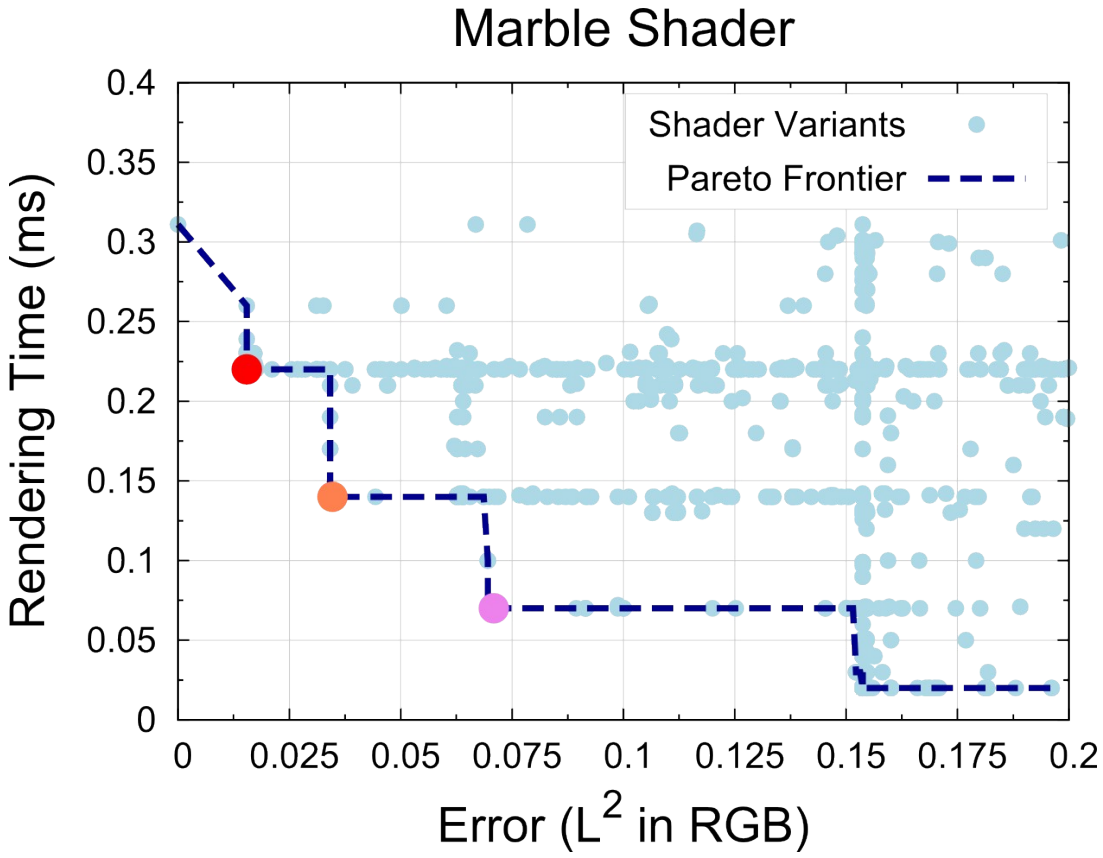
Trashcan shader

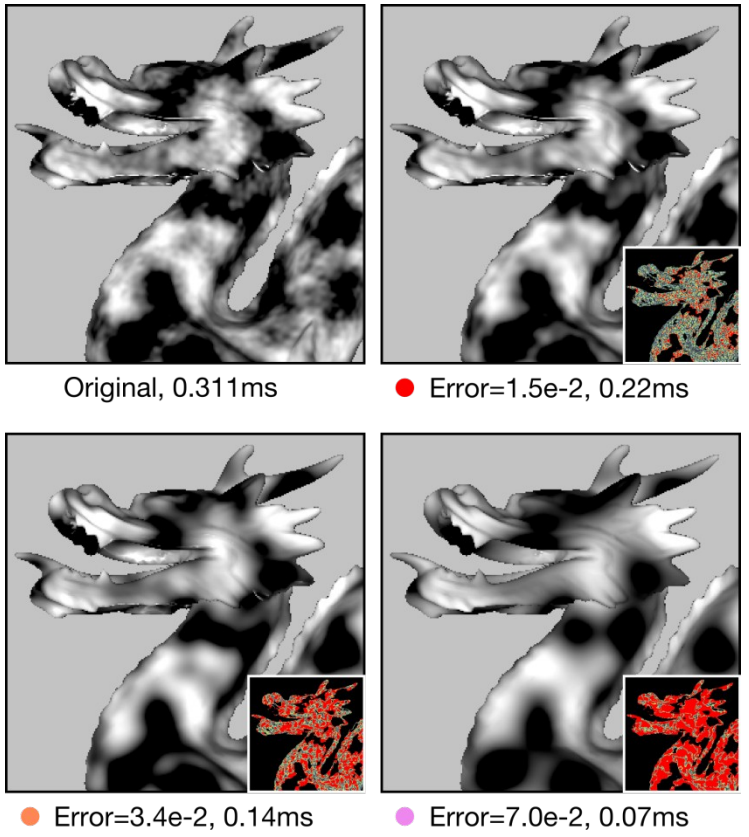
supersampled (25)
environment map
(15K triangles)

Human Head

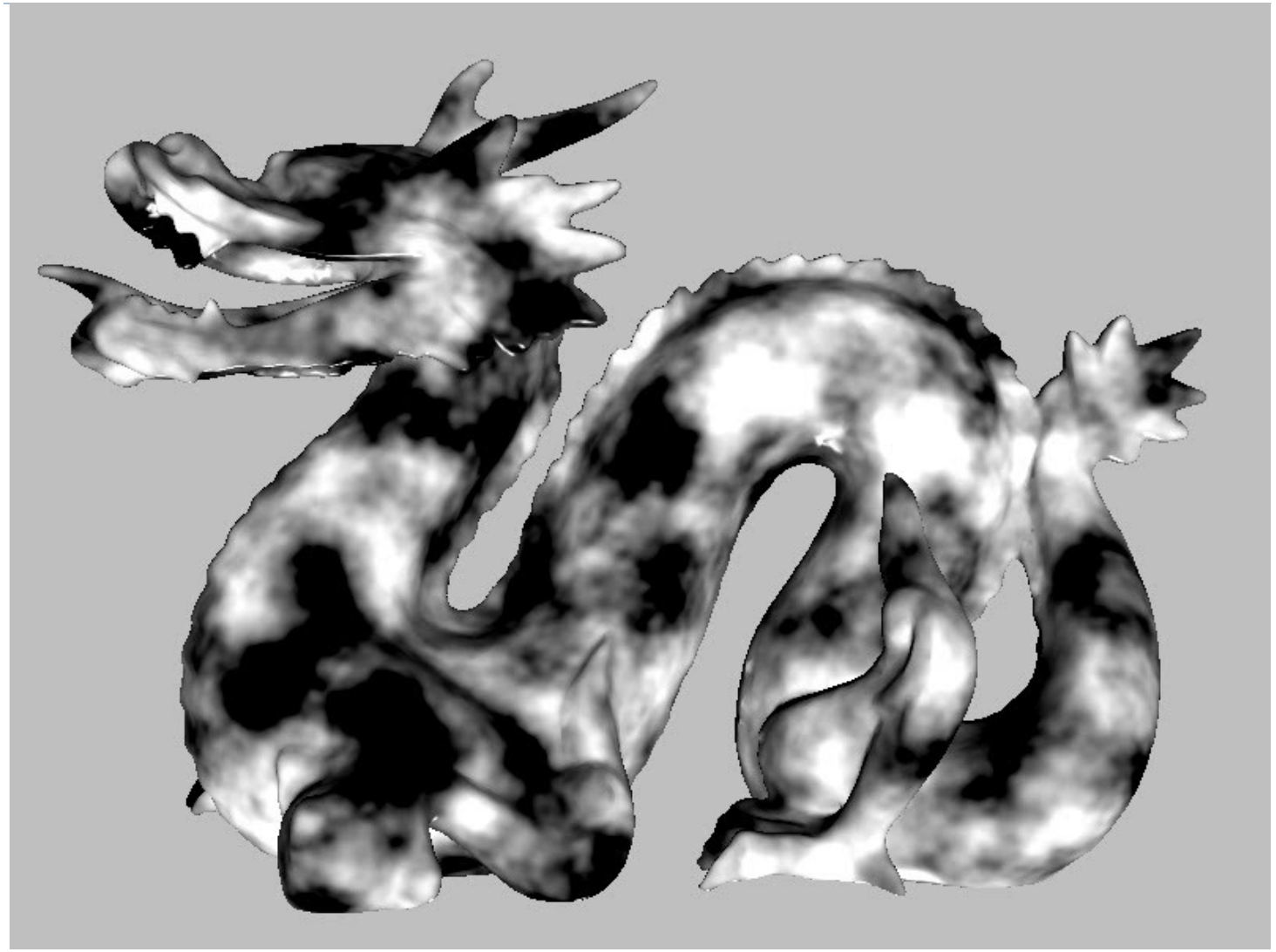
approximate subsurface
scattering of human skin
(300K triangles)

Marble Shader

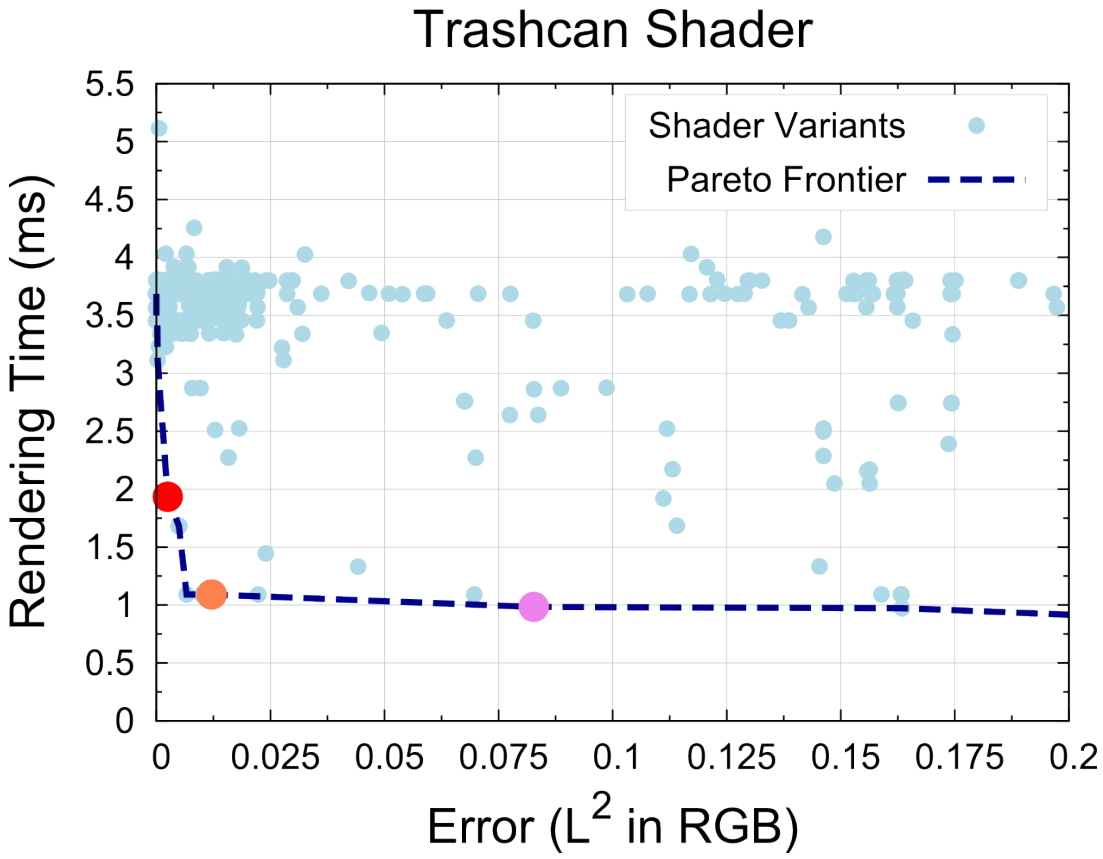


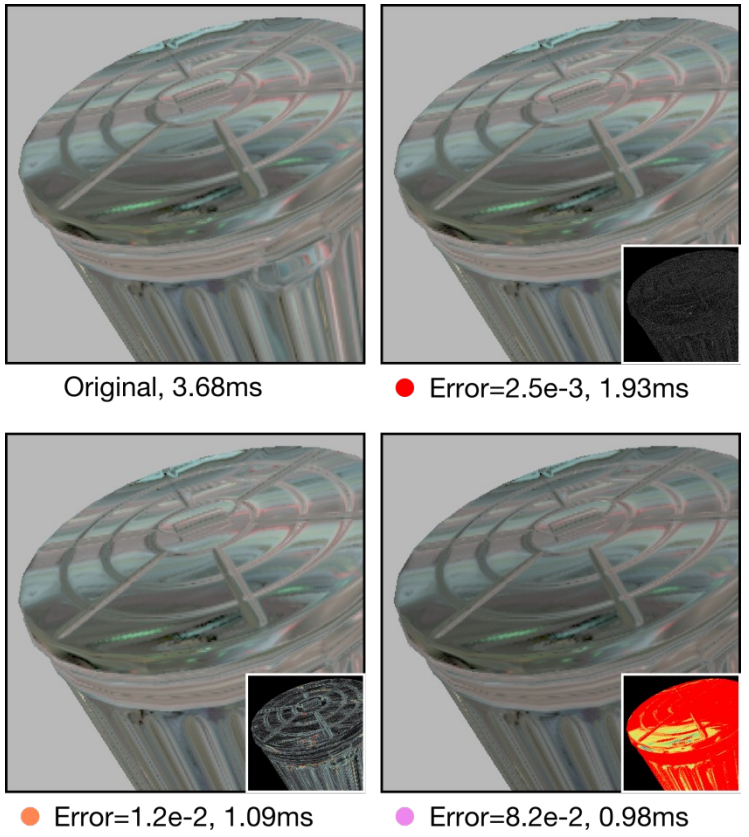


Marble Shader



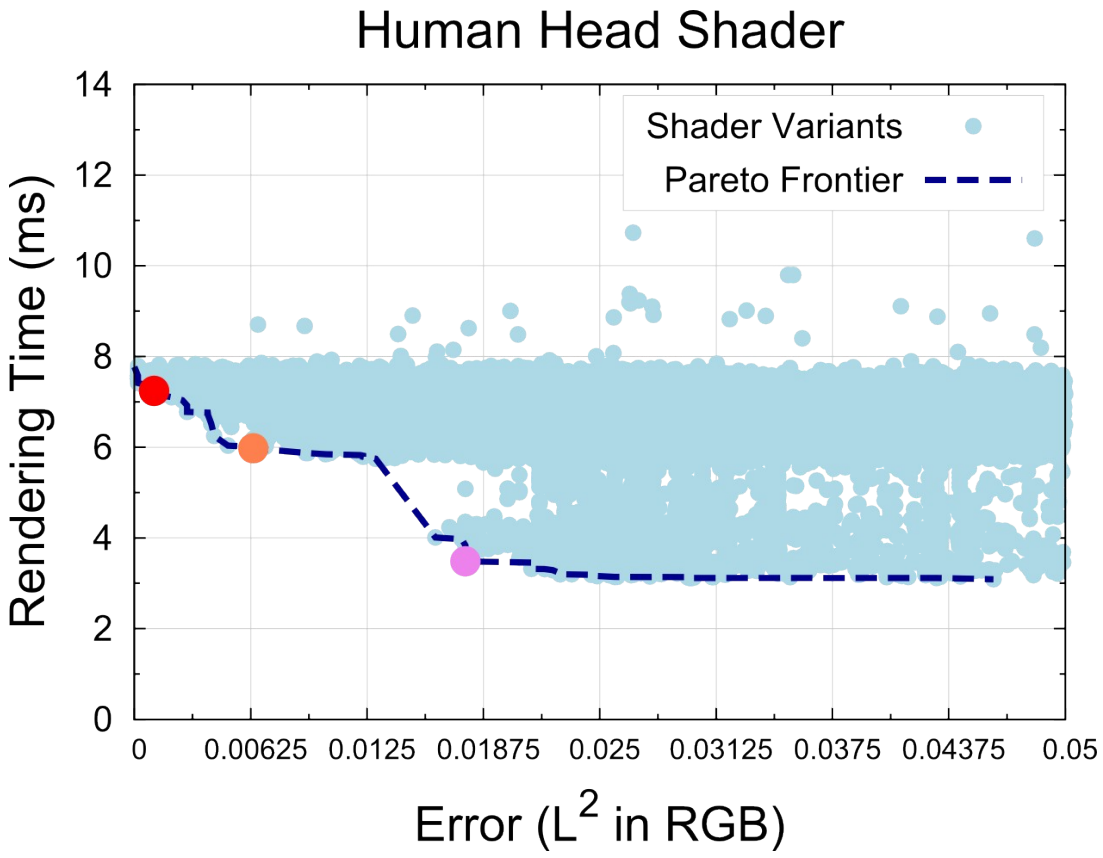
Trashcan Shader

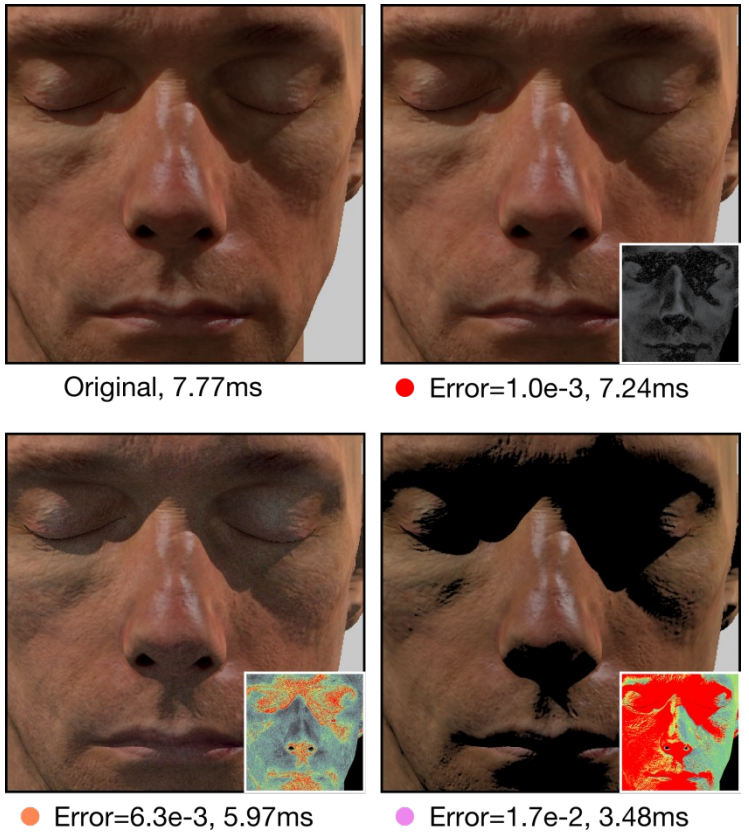




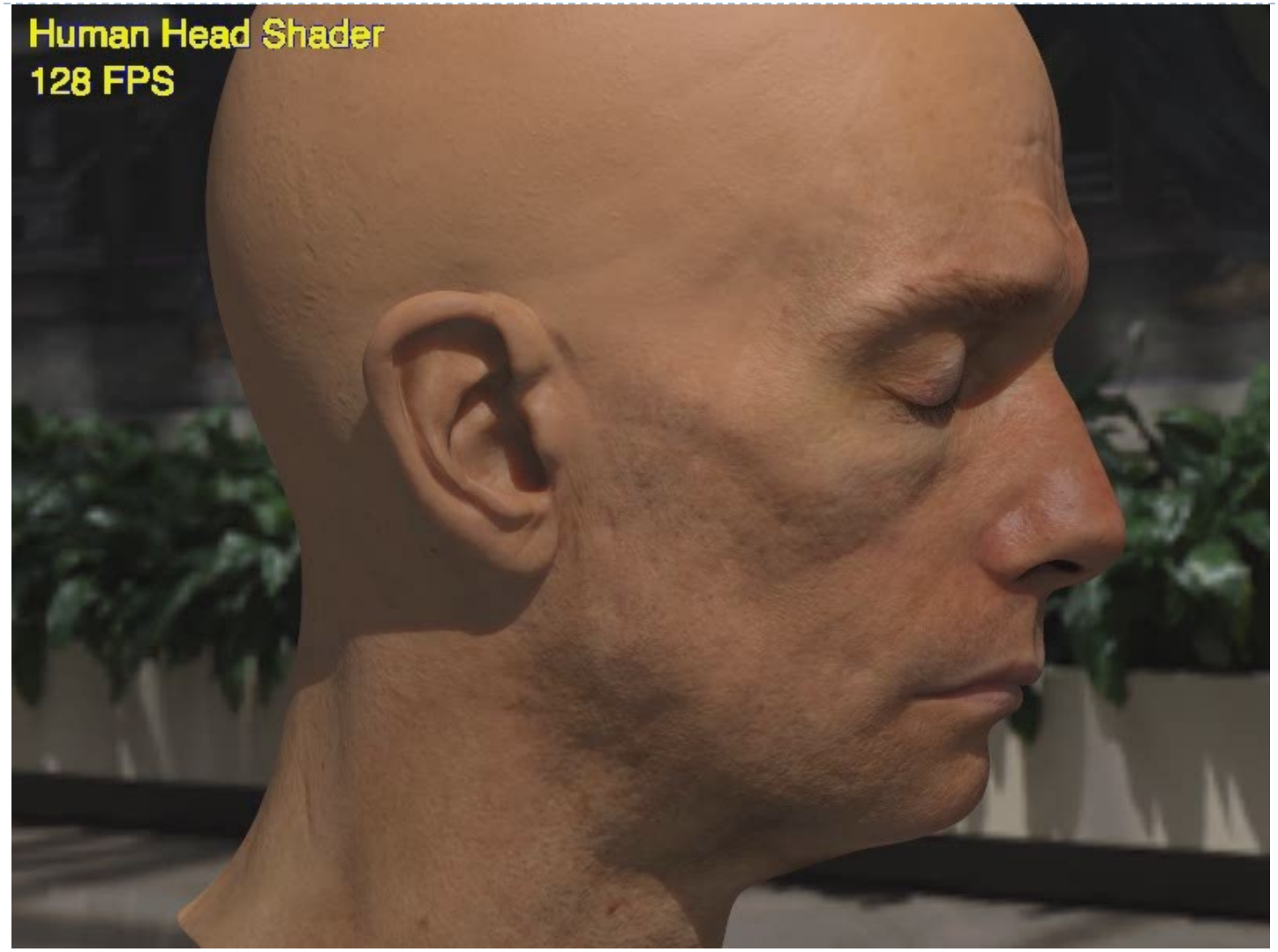
Trashcan Shader

Human Head Shader

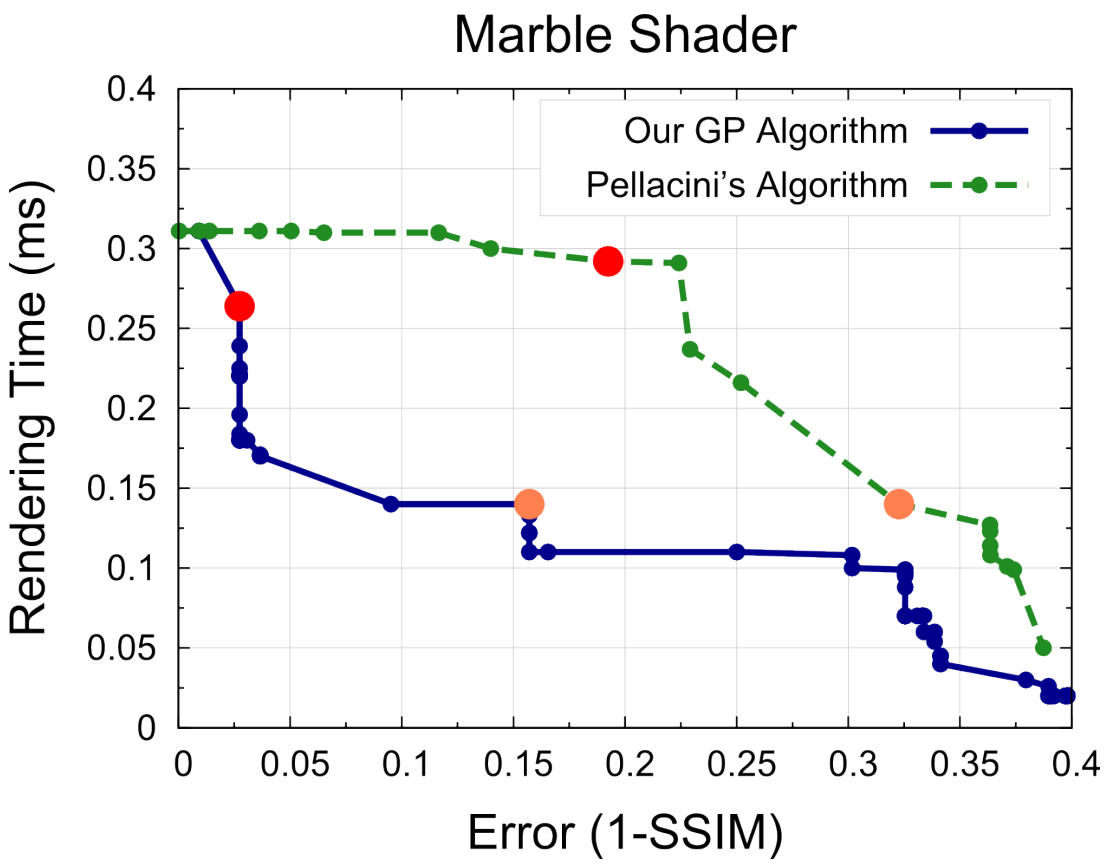


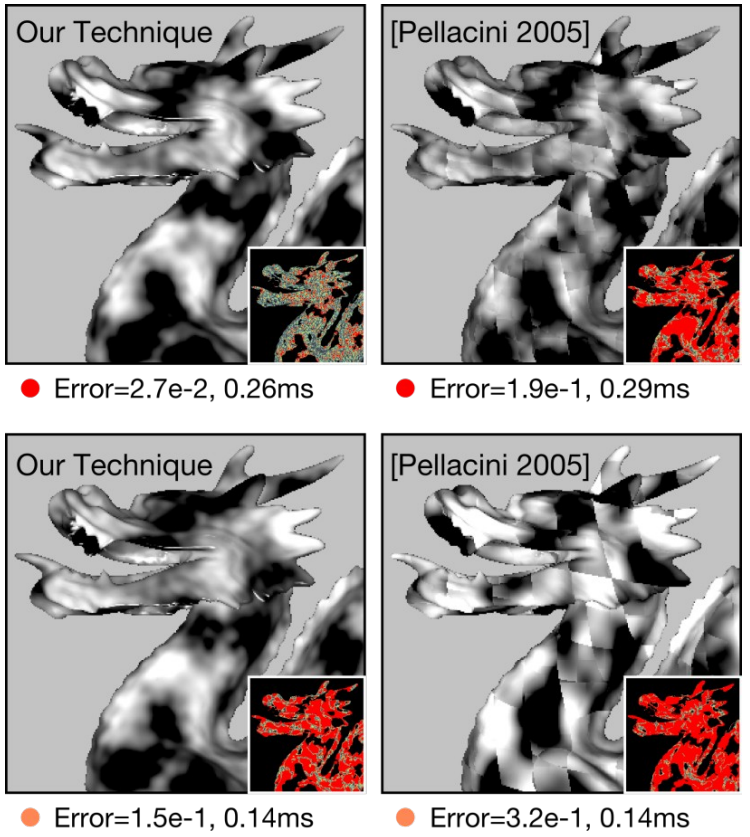


Human Head Shader



Marble Shader – SSIM & Previous Work





(Previous approaches are not well-founded on multi-pass shaders.)

Conclusion

- Graphics shader software simplification
  - “Continuous functions”
  - Efficiency is critical
  - Output consumed by humans
- Multi-objective genetic programming approach
  - NSGA-II plus mutation, crossover, tournament *k*, crowding heuristic
  - Rapid error and runtime (fitness) approximation
  - 2.5x better than previous work at constant error
  - Applies to multi-pass shaders