V8 Performance Characterization & Analysis (Octane.raytrace)

STO/SLOT

02/01/2016

Configuration

Hardware

- Intel®Core™ Processor i7 4790 @ 3.60 GHz, 4 Cores

Software

- Ubuntu 14.04.3 LTS
- Node.js 4.2.2

Benchmark

- Octane.RayTrace

Opportunities

- To reduce in-lining overhead
- To reduce memory allocations and increase code hoisting

Octane.raytrace

	Node.js - base
Elapsed time (ms)	70000
Standard Score	71817
Iterations	67936
Score (iterations/second)	970.51
Path Length:USER	**8,441,005**
Path Length:USER [JIT]	7,959,868
Path Length:USER [VM]	371,404
Path Length:node	0
metric_CPU utilization %	100%
metric_CPI	**0.45**

Octane.raytrace functional breakdown

Module / Function / Call Stack	CPU_CLK_UNHALTED.THREAD	INST_RETIRED.ANY	CPI Rate
[Dynamic code]	**91%**	**94.3%**	**0.42**
Flog::RayTracer::Shape::Sphere::intersect	16.7%	20.1%	0.36
Flog::RayTracer::Engine::rayTrace	15.1%	13.4%	0.491
Flog::RayTracer::Vector::normalize	9.9%	8.8%	0.49
Flog::RayTracer::Engine::testIntersection	8.2%	7.8%	0.46
Flog::RayTracer::Shape::Plane::intersect	5.0%	5.1%	0.43
Flog::RayTracer::Vector::subtract	3.2%	3.9%	0.36
Flog::RayTracer::Color::blend	2.5%	3.0%	0.35
Flog::RayTracer::Color::multiplyScalar	1.4%	0.9%	0.704
Flog::RayTracer::Color::add	1.3%	2.7%	0.213
MathPowStub	2.8%	2.2%	0.77
Node	**6.3%**	**4.4%**	**0.63**
V8::internal::Object::Equals	0.7%	1.0%	0.39
V8::internal::Runtime_Equals	0.4%	0.5%	0.39

Hand In-lining experiment – Opt1

```
Flog.RayTracer.Engine.prototype = {
…
  rayTrace: function(info, ray, scene, depth){
     var color = Flog.RayTracer.Color.prototype.multiplyScalar(info.color, scene.background.ambience);
      …
     for(var i=0; i<scene.lights.length; i++){
       var light = scene.lights[i];
        var v = Flog.RayTracer.Vector.prototype.subtract(light.position,
info.position).normalize();
        if(…….){
          var L = v.dot(info.normal);
color = Flog.RayTracer.Color.prototype.add(color,Flog.RayTracer.Color.prototype.multiply(
info.color, Flog.RayTracer.Color.prototype.multiplyScalar(light.color, L)));
            …
        }
       if(…..){   /* if condition */
          var Lv =
Flog.RayTracer.Vector.prototype.subtract(info.shape.position,light.position).normalize();
          var E =
Flog.RayTracer.Vector.prototype.subtract(scene.camera.position,info.shape.position).nor
malize();
         var H = Flog.RayTracer.Vector.prototype.subtract(E, Lv).normalize();
          color = Flog.RayTracer.Color.prototype.add(
                    Flog.RayTracer.Color.prototype.multiplyScalar(light.color,
glossWeight),color);
        }   /* End of if condition */
     }
  }
};
```

In-lined functions

```
Flog.RayTracer.Vector.prototype = {
   x : 0.0,
   y : 0.0,
   z : 0.0,
…
normalize : function() {
      var m = this.magnitude();
      return new Flog.RayTracer.Vector(this.x / m, this.y / m,
this.z / m);
   },
   magnitude : function() {
      return Math.sqrt((this.x * this.x) + (this.y * this.y) + (this.z
* this.z));
   },
subtract : function(v, w) {
      if(!w || !v) throw 'Vectors must be defined [' + v
            + ',' + w + ']';
      return new Flog.RayTracer.Vector(v.x - w.x, v.y - w.y, v.z -
w.z);
   },
 add : function(v, w) {
      return new Flog.RayTracer.Vector(w.x + v.x, w.y + v.y, w.z
+ v.z);
   },
 multiplyScalar : function(v, w) {
      return new Flog.RayTracer.Vector(v.x*w, v.y*w, v.z*w);
   }
 dot : function(w) {
      return this.x * w.x + this.y * w.y + this.z * w.z;
   },
};
```

V8 does inline all the highlighted functions from “Flog.RayTracer.Vector.prototype = {}” class but as shown on the next few slides hand in-lining seems to help more.

Hand In-lining experiment – Opt1

```
Flog.RayTracer.Engine.prototype = {
…
   rayTrace: function(info, ray, scene, depth){
      var color = Flog.RayTracer.Color.prototype.multiplyScalar(info.color, scene.background.ambience);
       …
      for(var i=0; i<scene.lights.length; i++){
        var light = scene.lights[i];
         var v = Flog.RayTracer.Vector.prototype.subtract(light.position,
info.position).normalize();
         if(…….){
           var L = v.dot(info.normal);
color = Flog.RayTracer.Color.prototype.add(color,Flog.RayTracer.Color.prototype.multiply(
info.color, Flog.RayTracer.Color.prototype.multiplyScalar(light.color, L)));
            …
         }
        if(…..){  /* if condition */
            var Lv =
Flog.RayTracer.Vector.prototype.subtract(info.shape.position,light.position).normalize();
            var E =
Flog.RayTracer.Vector.prototype.subtract(scene.camera.position,info.shape.position).nor
malize();
           var H = Flog.RayTracer.Vector.prototype.subtract(E, Lv).normalize();
            color = Flog.RayTracer.Color.prototype.add(
                      Flog.RayTracer.Color.prototype.multiplyScalar(light.color,
glossWeight),color);
         }   /* End of if condition */
      }
      color.limit();
       return color;
   }
};
```

In-lined functions

```
Flog.RayTracer.Color.prototype = {
    red : 0.0,
    green  : 0.0,
    blue  : 0.0,
…
add : function(c1, c2) {
  var result = new Flag.RayTracer.Color(0,0,0);
  result.red = c1.red + c2.red;
  result.green = c1.green + c2.green;
  result.blue = c1.blue + c2.blue;
},
multiply : function(c1, c2) {
  var result = new Flag.RayTracer.Color(0,0,0);
  result.red = c1.red * c2.red;
  result.green = c1.green * c2.green;
  result.blue = c1.blue * c2.blue;
},
multiplyScalar : function(c1, f) {
  var result = new Flag.RayTracer.Color(0,0,0);
  result.red = c1.red * f;
  result.green = c1.green * f;
  result.blue = c1.blue * f;
},
limit: function() {
this.red = (this.red > 0.0)? ((this.red > 1.0) ? 1.0: this.red) : 0.0;
this.green = (this.green > 0.0)? ((this.green > 1.0) ? 1.0:
this.green) : 0.0;
this.blue = (this.blue > 0.0)? ((this.blue > 1.0) ? 1.0: this.blue) :
0.0;
},
…
};
```

V8 does inline all the highlighted functions from “Flog.RayTracer.Color.prototype = {}” class but as shown on the next few slides hand in-lining seems to help more.

Idea: Memory allocation and code hoisting opportunities – Opt2

```
Flog.RayTracer.Color.prototype = {
   red : 0.0,    green : 0.0,    blue : 0.0,
   initialize : function(r, g, b) {
      if(!r) r = 0.0;
      if(!g) g = 0.0;
      if(!b) b = 0.0;
      this.red = r;
      this.green = g;
      this.blue = b;
   },
   add : function(c1, c2){
      var result = new Flog.RayTracer.Color(0,0,0);
      result.red = c1.red + c2.red;
      result.green = c1.green + c2.green;
      result.blue = c1.blue + c2.blue;
      return result;
   },
multiply : function(c1, c2) {
      var result = new Flog.RayTracer.Color(0,0,0);
      result.red = c1.red * c2.red;
      result.green = c1.green * c2.green;
      result.blue = c1.blue * c2.blue;
      return result;
   },

   multiplyScalar : function(c1, f) {
      var result = new Flog.RayTracer.Color(0,0,0);
      result.red = c1.red * f;
      result.green = c1.green * f;
      result.blue = c1.blue * f;
      return result;
   }
…
}
```

```
blend: function(c1, c2, w){
      var result = new Flog.RayTracer.Color(0,0,0);
      result = Flog.RayTracer.Color.prototype.add(
         Flog.RayTracer.Color.prototype.multiplyScalar(c1, 1 - w),
         Flog.RayTracer.Color.prototype.multiplyScalar(c2, w)
       );
      return result;
   },
```

Source code for the class Flog.RayTracer.Color();

Idea: Memory allocation and code hoisting opportunities – Opt2

```
Flog.RayTracer.Engine.prototype = {
var color =
Flog.RayTracer.Color.prototype.multiplyScalar(info.color,
scene.background.ambience);
…
 rayTrace: function(info, ray, scene, depth){
  for(var i=0; i<scene.lights.length; i++){
    ….
    if(…..){
     color = Flog.RayTracer.Color.prototype.add(
       color,Flog.RayTracer.Color.prototype.multiply(info.color,
      Flog.RayTracer.Color.prototype.multiplyScalar(light.color,
L  ) ) );
     }
    if (..) {
    color = Flog.RayTracer.Color.prototype.blend(color,
refl.color, info.shape.material.reflection);
  }
…
}
..
}
```

Step1- inline

```
→ /* multiplyScalar() */
var u_ms1 = new Flog.RayTracer.Color(0,0,0);
u_ms1.red     =  light.color.red * L;
u_ms1.green = light.color.green * L;
u_ms1.blue    = light.color.blue * L;

→ /* multiply() */
var u_ms2 = new Flog.RayTracer.Color(0,0,0);
u_ms2.red     = info.color.red * u_ms1.red;
u_ms2.green = info.color.green * u_ms1.green;
u_ms2.blue   = info.color.blue * u_ms1.blue;

→/* add() */
var u_color = new Flog.RayTracer.Color(0,0,0);
u_color..red    = color.red     + u_ms2.red;
u_color.green = color.green + u_ms2.green;
u_color.blue    = color.blue   + u_ms2.blue;
color = u_color;
```

Step 2

Line# 1061

```
color.red   = color.red +    (info.color.red * (light.color.red * L));
color.green = color.green + (info.color.green * (light.color.green * L));
color.blue    = color.blue +  (info.color.blue*  (light.color.blue * L));
```

Octane.raytrace functional breakdown after the change

Module / Function / Call Stack	CPU_CLK_UNHALTED.THREAD	INST_RETIRED.ANY	CPI Rate
[Dynamic code]	**91%**	**92.8%**	**0.56**
Flog::RayTracer::Engine::rayTrace	33.5%	28.4%	0.56
Flog::RayTracer::Shape::Sphere::intersect	19.5%	25.8%	0.36
Flog::RayTracer::Shape::Plane::intersect	5.1%	6.5%	0.37
Flog::RayTracer::Engine::testIntersection	4.7%	4.1%	0.54
MathPowStub	3.8%	1.8%	0.99
Flog::RayTracer::Vector::normalize	1.2%	0.7%	0.8
Node	**6.9%**	**5.5%**	**0.59**
V8::internal::Object::Equals	0.9%	1.5%	0.29
V8::internal::Runtime_Equals	0.4%	0.6%	0.35

Octane.raytrace

	Base	modified	Mod/base
Elapsed time (ms)	70000	70000	1.00
Standard Score	71817	88,376	1.23
Iterations	67936	83600	1.23
Score (iterations/second)	970.51	1194.29	1.23
Path Length:USER	**8,441,005**	**6,413,436**	**0.76**
Path Length:USER [JIT]	7,959,868	5,951,668	0.75
Path Length:USER [VM]	371,404	352,739	0.95
Path Length:node	0	0	
metric_CPU utilization %	100%	100%	
metric_CPI	**0.45**	**0.48**	**1.07**

Overall, hand in-lining improved total score by ~23%

Currently there are issues in matching JavaScript source code the jitte'd code while using VTune.

Questions?

intel®

Software