Sphere tracing and all that

Thomas Strathmann

October 12, 2009

TODO: describe algorithm via pseudo code, highlight computational issues make prettier (and more importantly!) more accurate drawings

1 Distance Fields

Given an implicit surface ∂S with $S \subseteq \mathbb{R}^3$ the corresponding distance field (also called signed distance function) $D : \mathbb{R}^3 \to \mathbb{R}$ which maps a point p from the domain of the distance field $\mathbb{R}^3$ to a real-valued distance value is obtained by distance transforming the surface using the formula

$$D(p) = \text{sign}(p) \cdot \min\{d(p, q) : q \in S\} \tag{1}$$

The metric $d(p, q)$ used is the standard Euclidean metric $\|p-q\|_2$. The expression $\text{sign}(p)$ indicates whether p is an interior or exterior point of S.

$$\text{sign}(p) = \begin{cases} -1 & \text{if } p \text{ is inside} \\ 1 & \text{if } p \text{ is outside} \end{cases} \tag{2}$$

1.1 Combining Distance Fields

Let $D_1, \ldots, D_N$ be distance fields with the same domain. Then the the union of the distance fields is given by

$$D_{union}(p) = \min\{D_1(p), \ldots, D_N(p)\} \tag{3}$$

Replacing the min operator with the max operator yields the intersection of the distance fields.

$$D_{intersection}(p) = \max\{D_1(p), \ldots, D_N(p)\} \tag{4}$$

1.2 Distance Field Transformations

A transformation T that acts upon the implicit surface S reprented by distance field D can be applied to the distance field like this

$$D'(p) = D(T^{-1}p) \tag{5}$$

As T is given as a homogenous matrix deriving D' is straightforward.

1.2.1 Translation

$$\text{Translate}(p, (x\ y\ z)^T) = \begin{pmatrix} 1 & 0 & 0 & x \\ 0 & 1 & 0 & y \\ 0 & 0 & 1 & z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{6}$$

$$\rightsquigarrow D'(p) = D(p - (x\ y\ z)^T) \tag{7}$$

1.2.2 Rotation

Rotation by an angle of α radians about one of the three axes is achieved with the following distance field transformations. In the following equations let $p = (x\ y\ z)$.

$$\text{RotateX}(p, \alpha) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{8}$$

$$D'(p) = D\left((x\ \ (\cos(\alpha) \cdot y + \sin(\alpha) \cdot y)\ \ (-\sin(\alpha) \cdot z + \cos(\alpha) \cdot z))^T\right) \tag{9}$$

$$\text{RotateY}(p, \alpha) = \begin{pmatrix} \cos(\alpha) & 0 & \sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{10}$$

$$D'(p) = D\left(((\cos(\alpha) \cdot x - \sin(\alpha) \cdot x)\ \ y\ \ (\sin(\alpha) \cdot z + \cos(\alpha) \cdot z))^T\right) \tag{11}$$

$$\text{RotateZ}(p, \alpha) = \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{12}$$

$$D'(p) = D\left(((\cos(\alpha) \cdot x + \sin(\alpha) \cdot x)\ \ (-\sin(\alpha) \cdot y + \cos(\alpha) \cdot y)\ \ z)^T\right) \tag{13}$$

1.3 Domain Transformations

TODO: distortion of object

1.4 Distance fields for basic objects

TODO: plane, sphere, torus, cylinder, etc.

1.5 Implementing Distance Fields

Distance fields can be implemented in several ways according to the constraints imposed by the given problem. The offline approach calculates a discrete approximation of the true distance field before the actual rendering step. Such

a method may store the distance values in a 3D texture for later use by the ray marching routine. The online approach calculates only the distance values needed during ray marching. In this setting the distance field is implemented as a function which is called by the ray marching routine.

2 Ray Marching

In contrast to ray tracing which solves the equations describing ray-object intersections analytically, ray marching evaluates the ray equation at discrete steps and tests for intersections numerically. Thus, ray marching can be used to render surfaces for which no analytical ray-object intersection equation (or solution) exist.

Let $v_0 \in \mathbb{R}^3$ be the origin of the (primary) ray. The direction is given by a unit vector v_d. The resulting ray equation is

$$ray(t) = v_0 + t \cdot v_d \tag{14}$$

In order to reduce the number of steps taken for each ray the step size t can be adapted using the distance field. Consider a point $p_1 = ray(t_1)$ and the corresponding distance field value $d_1 = D(p)$. If d_1 is positive, it denotes the distance to the nearest object. The optimal step size t_2 would then reduce the distance to 0, thus placing the new point exactly on the surface of the nearest object. This is trivially achieved by computing the new point p_2 with a step size $t_2 = d_1$. In the case that d_1 is negative, p_1 is inside the object. If the object is not translucent and no transmission effects need to be computed the algorithm proceeds to cast the next primary ray. Additionally a maximum length ray_{max} of the ray is imposed as a termination condition.

This method was introduced by Hart in [Har94] under the name "sphere tracing". The reason for this name becomes clear when considering the two rays cast in figure 1 and 2 respectively. There the ray originates from a point v_0 and is traced for several steps. A problem of this approach is illustrated in figure 2. The number of steps and thus distance field evaluations dramatically increases if the ray passes in close proximity of an object without intersection. To mitigate this problem the test whether the ray intersects with an object (i.e. reaches the inside of the volume and the distance field value is negative) can be changed such that any distance value less than a small positive constant d_{max} is considered as negative and thus lying inside the volume.

2.1 Computing the Primary Ray

In ray-x methods of rendering the primary ray describes the ray originating from a point of view (POV, camera) and passing through a pixel (x, y) of the image plane. The primary ray is computed according to the chosen view geometry which includes the width, height and origin of the image plane. The example GLSL code fragmet given below calculates a ray originating from the point $p = (0\ 0\ -2)^T$ (assuming a coordinate system with the camera looking down the negative z-axis) and passing through the "pixel" v_0. The vector v_0 is computed such that the image plane coincides with the x-y-plane of the coordinate system and ranges from $(-1,\ -1)$ at the lower left corner to $(1,\ 1)$ at the upper right with the origina 0 being in the center of the image plane.

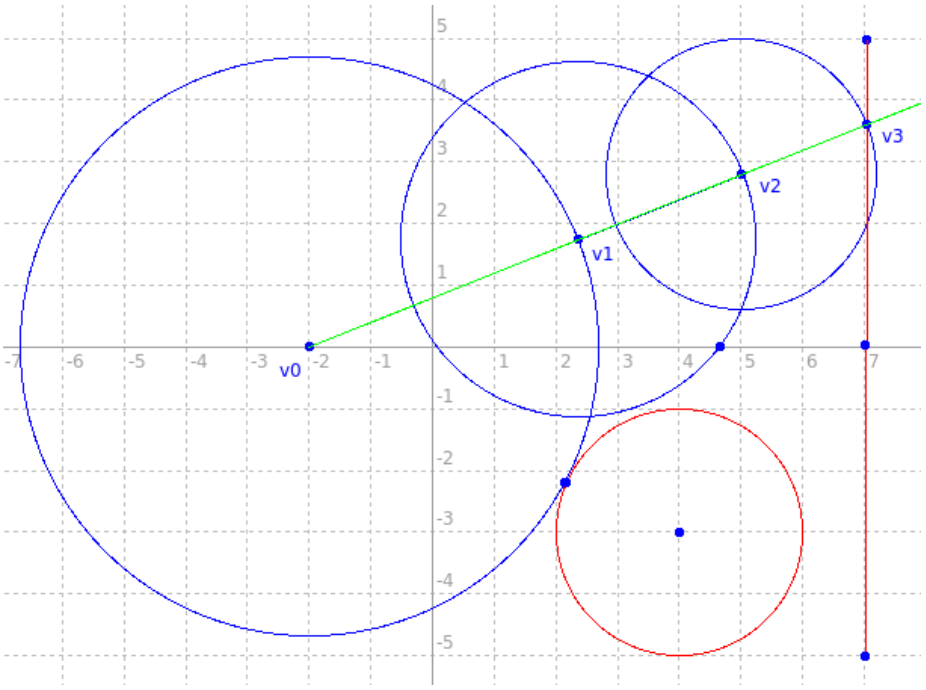

Figure 1: Tracing a ray in a 2D scene consisting of a circle and a straight line

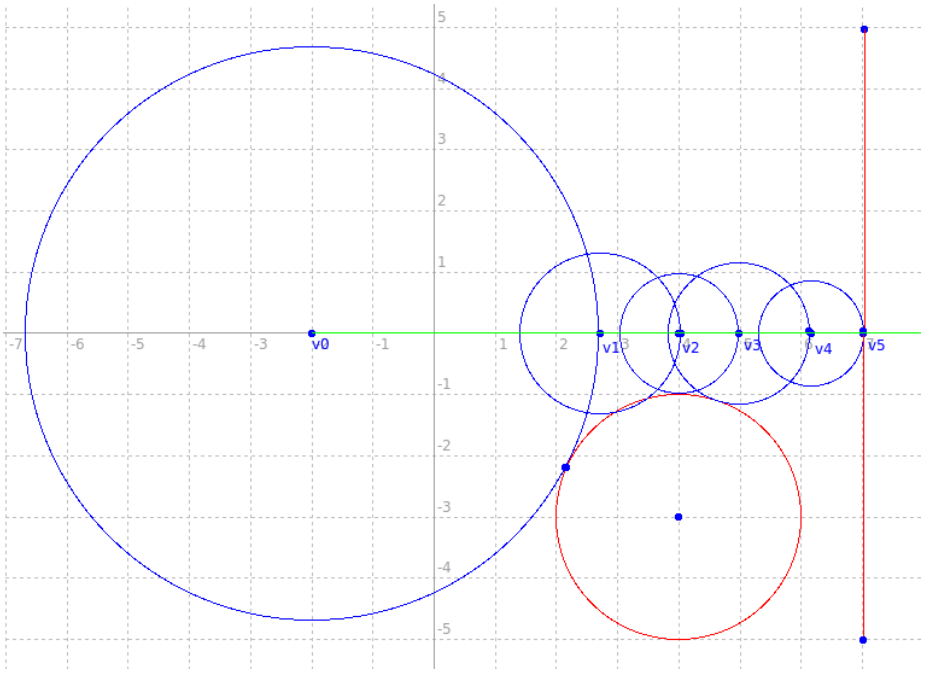

Figure 2: Demonstration of the problem near volume boundaries

```
uniform float width, height;

vec3 p = vec3(0, 0, -2);
vec3 v0 = vec3((gl_FragCoord.x - 0.5*width) / width,
               (gl_FragCoord.y - 0.5*height) / height,
               0.0);
vec3 vd = normalize(v - p);

vec3 ray(vec3 v0, vec3 vd, float t) {
  return v0 + t * vd;
}
```

2.2 Shading

The surface normal needed for shading is given by the gradient of the distance field. The partial derivative is computed numerically using central difference.

$$\nabla p \approx \begin{pmatrix} D(p+(\delta,0,0)) - D(p-(\delta,0,0)) \\ D(p+(0,\delta,0)) - D(p-(0,\delta,0)) \\ D(p+(0,0,\delta)) - D(p-(0,0,\delta)) \end{pmatrix} \tag{15}$$

The surface normal n then is

$$n = \frac{\nabla p}{\|\nabla p\|} \tag{16}$$

Once the normal is computed via this method a traditional shading model like (Blinn-)Phong can be used to determine diffuse and specular components at the intersection point given the position of a (point) light source *light*.

2.3 Shadows

Adding shadows is in theory trivial once the distance field and sphere tracing code is in place. Computing the a value $shadow \in [0,\ 1]$ where 1 means the point is completely occluded involves tracing a secondary ray from the intersection point to the light source.

TODO:

- number of steps
- (adaptive) step size
- averaging the resulting distance values to obtain *shadow*

References

[Har93] John C. Hart. Ray tracing implicit surfaces. In *Siggraph 93 Course Notes: Design, Visualization and Animation of Implicit Surfaces*, pages 1–16, 1993.

[Har94] John C. Hart. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12:527–545, 1994.

[Kno07] Aaron Knoll. A survey of implicit surface rendering methods, and a proposal for a common sampling framework. In Hans Hagen, Martin Hering-Bertram, and Christoph Garth, editors, *Visualization of Large and Unstructured Data Sets*, volume S-7 of *LNI*, pages 164–177. GI, 2007.

[Qui08] Iñigo Quilez. Rendering worlds with two triangles with raytracing on the gpu in 4096 bytes, August 2008.