

4.1 Basic Shape Interface

(a). // constructor :

```
Shape::Shape(const Transform *ObjectToWorld,
             const Transform *WorldToObject, bool reverseOrientation)
    : ObjectToWorld(ObjectToWorld), WorldToObject(WorldToObject),
      reverseOrientation(reverseOrientation),
      transformSwapsHandedness(ObjectToWorld->SwapsHandedness()) {
}
```

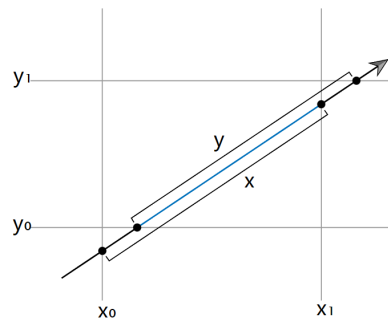
(b). // return bounding box

(a). ~~Shape Interface~~
virtual Bounds3f ObjectBound() const = 0;

(b) <<Shape Method Definitions>> += ▲ ▼
Bounds3f Shape::WorldBound() const {
 return (*ObjectToWorld)(ObjectBound());
}

(c). Ray - Bounds Intersections:

(i). general cases:



```
template <typename T>
inline bool Bounds3<T>::IntersectP(const Ray &ray, Float *hitt0,
                                   Float *hitt1) const {
    Float t0 = 0, t1 = ray.tMax;
    for (int i = 0; i < 3; ++i) {
        <<Update interval for ith bounding box slab>>
    }
    if (hitt0) *hitt0 = t0;
    if (hitt1) *hitt1 = t1;
    return true;
}
```

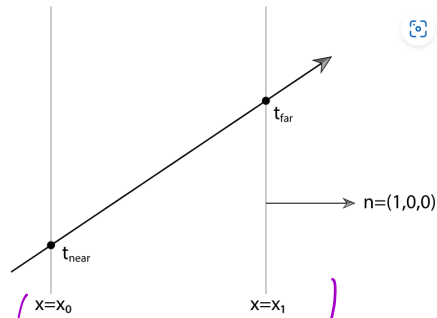
$$0 = a(o_x + t\mathbf{d}_x) + b(o_y + t\mathbf{d}_y) + c(o_z + t\mathbf{d}_z) + d$$

$$= (a, b, c) \cdot \mathbf{o} + t(a, b, c) \cdot \mathbf{d} + d.$$



$$t = \frac{-d - (a, b, c) \cdot \mathbf{o}}{(a, b, c) \cdot \mathbf{d}}$$

(ii) . special case (in practical case)



$$\begin{cases} \vec{y} = \vec{o} + t \cdot \vec{d} \\ ax + by + cz + d = 0 \end{cases}$$

$$ax + by + cz + d = 0$$

$$x - x_0 = 0$$

$$\begin{cases} a=1 \\ b=0 \\ c=0 \\ d=-x_0 \end{cases}$$

$$x - x_1 = 0$$

$$\begin{cases} a=1 \\ b=0 \\ c=0 \\ d=-x_1 \end{cases}$$

$$t = \frac{-d - (a, b, c) \cdot \vec{o}}{(a, b, c) \cdot \vec{d}}$$

$$t = \frac{x_t - o_x}{d_x}$$

<<Update interval for ith bounding box slab>>=

```
Float invRayDir = 1 / ray.d[i];
Float tNear = (pMin[i] - ray.o[i]) * invRayDir;
Float tFar = (pMax[i] - ray.o[i]) * invRayDir;
```

<<Update parametric interval from slab intersection t values>> ⊞



```
template <typename T>
inline bool Bounds3<T>::IntersectP(const Ray &ray, Float *hitt0,
    Float *hitt1) const {
    Float t0 = 0, t1 = ray.tMax;
    for (int i = 0; i < 3; ++i) {
        <<Update interval for ith bounding box slab>> ⊞
        Float invRayDir = 1 / ray.d[i];
        Float tNear = (pMin[i] - ray.o[i]) * invRayDir;
        Float tFar = (pMax[i] - ray.o[i]) * invRayDir;
        <<Update parametric interval from slab intersection t values>> ⊞
        if (tNear > tFar) std::swap(tNear, tFar);
        <<Update tFar to ensure robust ray-bounds intersection>> ⊞
        t0 = tNear > t0 ? tNear : t0;
        t1 = tFar < t1 ? tFar : t1;
        if (t0 > t1) return false;
    }
    if (hitt0) *hitt0 = t0;
    if (hitt1) *hitt1 = t1;
    return true;
}
```


(d). Intersection Tests:

```
<<Shape Interface>>+=▲▼
```

```
virtual bool Intersect(const Ray &ray, Float *tHit,  
    SurfaceInteraction *isect, bool testAlphaTexture = true) const = 0;
```

There are a few important things to keep in mind when reading (and writing) intersection routines:

- The `Ray` structure contains a `Ray::tMax` member that defines the endpoint of the ray. Intersection routines must ignore any intersections that occur after this point.
- If an intersection is found, its parametric distance along the ray should be stored in the `tHit` pointer that is passed into the intersection routine. If there are multiple intersections along the ray, the closest one should be reported.
- Information about an intersection is stored in the `SurfaceInteraction` structure, which completely captures the local geometric properties of a surface. This class is used heavily throughout pbrt, and it serves to cleanly isolate the geometric portion of the ray tracer from the shading and illumination portions. The `SurfaceInteraction` class was defined in Section 2.10.[†]
- The rays passed into intersection routines are in world space, so shapes are responsible for transforming them to object space if needed for intersection tests. The intersection information returned should be in world space.



```
virtual bool IntersectP(const Ray &ray,  
    bool testAlphaTexture = true) const {  
    Float tHit = ray.tMax;  
    SurfaceInteraction isect;  
    return Intersect(ray, &tHit, &isect, testAlphaTexture);  
}
```

Recall

```
SurfaceInteraction::SurfaceInteraction(const Point3f &p,  
    const Vector3f &pError, const Point2f &uv, const Vector3f &wo,  
    const Vector3f &dpdu, const Vector3f &dpdv,  
    const Normal3f &dndu, const Normal3f &dndv,  
    Float time, const Shape *shape)  
: Interaction(p, Normal3f(Normalize(Cross(dpdu, dpdv))), pError, wo,  
    time, nullptr),  
    uv(uv), dpdu(dpdu), dpdv(dpdv), dndu(dndu), dndv(dndv),  
    shape(shape) {  
    <<Initialize shading geometry from true geometry>> +  
    shading.n = n;  
    shading.dpdu = dpdu;  
    shading.dpdv = dpdv;  
    shading.dndu = dndu;  
    shading.dndv = dndv;  
  
    <<Adjust normal based on orientation and handedness>> +  
    if (shape && (shape->reverseOrientation ^  
        shape->transformSwapsHandedness)) {  
        n *= -1;  
        shading.n *= -1;  
    }  
}
```

4.2. Sphere.

(a).

Surface $\left\{ \begin{array}{l} \text{implicit} \\ \text{parametric} \end{array} \right.$

$f(x, y, z) = 0$ e.g. $x^2 + y^2 + z^2 - 1 = 0$

$$\begin{cases} x = r \sin \theta \cos \phi \\ y = r \sin \theta \sin \phi \\ z = r \cos \theta \end{cases}$$

$(\theta, \phi) \in \underbrace{(0, \pi)}_{(0, 2\pi)}$



$(u, v) \in [0, 1]^2$

$\begin{cases} \phi = u \phi_{\max} \\ \theta = \theta_{\min} + v (\theta_{\max} - \theta_{\min}) \end{cases}$

Texture mapping $(u, v) \in [0, 1]^2 \longrightarrow \text{sphere } (x, y, z)$

(b). Bounding

```
Bounds3f Sphere::ObjectBound() const {
    return Bounds3f(Point3f(-radius, -radius, zMin),
                    Point3f(radius, radius, zMax));
}
```

(c). Intersection Tests

$$x^2 + y^2 + z^2 - r^2 = 0.$$

$$(o_x + t \mathbf{d}_x)^2 + (o_y + t \mathbf{d}_y)^2 + (o_z + t \mathbf{d}_z)^2 = r^2.$$

↑
unknown

$$at^2 + bt + c = 0,$$

$$a = \mathbf{d}_x^2 + \mathbf{d}_y^2 + \mathbf{d}_z^2$$

$$b = 2(\mathbf{d}_x \mathbf{o}_x + \mathbf{d}_y \mathbf{o}_y + \mathbf{d}_z \mathbf{o}_z)$$

$$c = \mathbf{o}_x^2 + \mathbf{o}_y^2 + \mathbf{o}_z^2 - r^2.$$

```

EFloat ox(ray.o.x, oErr.x), oy(ray.o.y, oErr.y), oz(ray.o.z, oErr.z);
EFloat dx(ray.d.x, dErr.x), dy(ray.d.y, dErr.y), dz(ray.d.z, dErr.z);

```



```

EFloat a = dx * dx + dy * dy + dz * dz;
EFloat b = 2 * (dx * ox + dy * oy + dz * oz);
EFloat c = ox * ox + oy * oy + oz * oz - EFloat(radius) * EFloat(radius);

```

<<Compute quadratic sphere coefficients>>

<<Solve quadratic equation for t values>>

```

EFloat t0, t1;
if (!Quadratic(a, b, c, &t0, &t1))
    return false;
<<Check quadric shape t0 and t1 for nearest intersection>> 
if (t0.UpperBound() > ray.tMax || t1.LowerBound() <= 0)
    return false;
EFloat tShapeHit = t0;
if (tShapeHit.LowerBound() <= 0) {
    tShapeHit = t1;
    if (tShapeHit.UpperBound() > ray.tMax)
        return false;
}

```

check(t)

0

t₀
low upper

t₁

<<Compute sphere hit position and φ>>

pHit

```

pHit = ray((Float)tShapeHit);
<<Refine sphere intersection point>> 
if (pHit.x == 0 && pHit.y == 0) pHit.x = 1e-5f * radius;
phi = std::atan2(pHit.y, pHit.x);
if (phi < 0) phi += 2 * Pi;

```

<<Test sphere intersection against clipping parameters>>

check(pHit)

```

if ((zMin > -radius && pHit.z < zMin) ||
    (zMax < radius && pHit.z > zMax) || phi > phiMax) {
    if (tShapeHit == t1) return false;
    if (t1.UpperBound() > ray.tMax) return false;
    tShapeHit = t1;
    <<Compute sphere hit position and φ>> 
    pHit = ray((Float)tShapeHit);
    <<Refine sphere intersection point>> 
    if (pHit.x == 0 && pHit.y == 0) pHit.x = 1e-5f * radius;
    phi = std::atan2(pHit.y, pHit.x);
    if (phi < 0) phi += 2 * Pi;
}

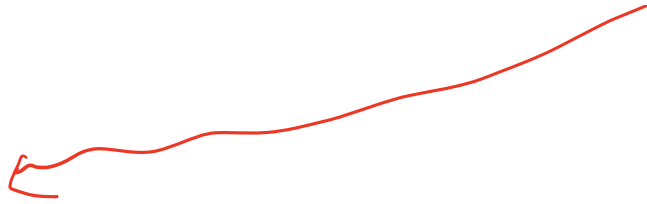
```

```

if ((zMin > -radius && pHit.z < zMin) ||
    (zMax < radius && pHit.z > zMax) || phi > phiMax)
    return false;
}

```

(d)



$$x = r \sin \theta \cos \phi$$

$$\begin{aligned} \frac{\partial p_x}{\partial u} &= \frac{\partial}{\partial u} (r \sin \theta \cos \phi) \\ &= r \sin \theta \frac{\partial}{\partial u} (\cos \phi) \\ &= r \sin \theta (-\phi_{\max} \sin \phi). \end{aligned}$$

$$\left(\begin{array}{l} p_x = r \sin \theta \cos \phi \\ p_y = r \sin \theta \sin \phi \\ z = r \cos \theta \end{array} \right)$$

$$\frac{\partial p_x}{\partial u} = -\phi_{\max} y.$$

$$\phi = u \phi_{\max}$$

$$\theta = \theta_{\min} + v (\theta_{\max} - \theta_{\min})$$

$$\frac{\partial p_y}{\partial u} = \phi_{\max} x,$$

$$\frac{\partial p_z}{\partial u} = 0.$$



$$\frac{\partial \mathbf{p}}{\partial u} = (-\phi_{\max} y, \phi_{\max} x, 0)$$

$$\frac{\partial \mathbf{p}}{\partial v} = (\theta_{\max} - \theta_{\min})(z \cos \phi, z \sin \phi, -r \sin \theta).$$

```
Float zRadius = std::sqrt(pHit.x * pHit.x + pHit.y * pHit.y);
Float invZRadius = 1 / zRadius;
Float cosPhi = pHit.x * invZRadius;
Float sinPhi = pHit.y * invZRadius;
Vector3f dpdu(-phiMax * pHit.y, phiMax * pHit.x, 0);
Vector3f dpdv = (thetaMax - thetaMin) *
    Vector3f(pHit.z * cosPhi, pHit.z * sinPhi,
        -radius * std::sin(theta));
```

$t_{\text{ShapeHit}} \rightarrow (t)$ $p_{\text{Hit}} \rightarrow (P)$

Surface Interaction *isect : : Interaction

Point 2 UV
 (Vec3f) dpdu dpdv
 (Normalf) dndu dndv
 (Shape) *shape

P
 n
 wo. (negative van div
 time

(a). `<<Sphere Method Definitions>>+=▲▼`

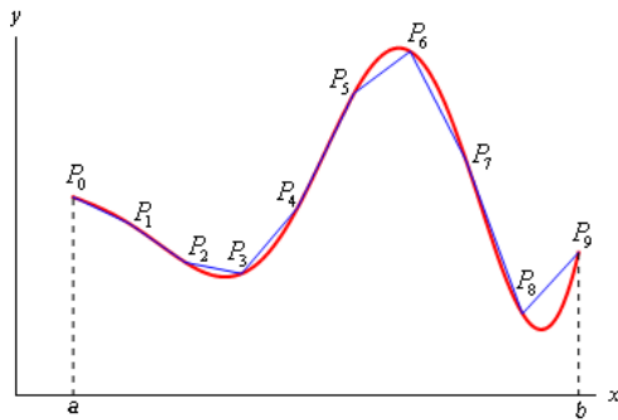
```
bool Sphere::Intersect(const Ray &r, Float *tHit,
    SurfaceInteraction *isect, bool testAlphaTexture) const {
    Float phi;
    Point3f pHit;
    <<Transform Ray to object space>> ⊕
    <<Compute quadratic sphere coefficients>> ⊕
    <<Solve quadratic equation for t values>> ⊕
    <<Compute sphere hit position and φ>> ⊕
    <<Test sphere intersection against clipping parameters>> ⊕
    <<Find parametric representation of sphere hit>> ⊕
    <<Compute error bounds for sphere intersection>> ⊕
    <<Initialize SurfaceInteraction from parametric information>> ⊕
    <<Update tHit for quadric intersection>> ⊕
        *tHit = (Float)tShapeHit;

    return true;
}
```

(b). `<<Sphere Method Definitions>>+=▲▼`

```
bool Sphere::IntersectP(const Ray &r, bool testAlphaTexture) const {
    Float phi;
    Point3f pHit;
    <<Transform Ray to object space>> ⊕
    <<Compute quadratic sphere coefficients>> ⊕
    <<Solve quadratic equation for t values>> ⊕
    <<Compute sphere hit position and φ>> ⊕
    <<Test sphere intersection against clipping parameters>> ⊕
    return true;
}
```

3-2.5 Surface Area



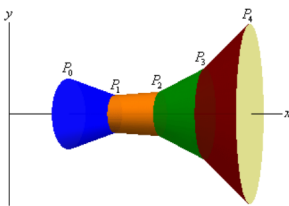
$$\begin{aligned} |P_{i-1} P_i| &= \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \\ &= \sqrt{\Delta x^2 + \Delta y^2} \end{aligned}$$

recall : mean value theorem

$$\exists x_i^* \in (x_{i-1}, x_i) \quad f(x_i) - f(x_{i-1}) = f'(x_i^*) (x_i - x_{i-1})$$

$$\Rightarrow \Delta y_i = f'(x_i^*) \Delta x$$

$$\begin{aligned} \therefore |P_{i-1} P_i| &= \sqrt{\Delta x^2 + [f'(x_i^*) \Delta x]^2} \\ &= \sqrt{1 + [f'(x_i^*)]^2} \cdot \Delta x \end{aligned}$$



The approximation on each interval gives a distinct portion of the solid and to make this clear each portion is colored differently. Each of these portions are called **frustums** and we know how to find the surface area of frustums.

The surface area of a frustum is given by,

$$A = 2\pi r l$$

where,

$$r = \frac{1}{2}(r_1 + r_2) \quad \begin{array}{l} r_1 = \text{radius of right end} \\ r_2 = \text{radius of left end} \end{array}$$

and l is the length of the slant of the frustum.

For the frustum on the interval $[x_{i-1}, x_i]$ we have,

$$\begin{aligned} r_1 &= f(x_i) \\ r_2 &= f(x_{i-1}) \\ l &= |P_{i-1} P_i| \quad (\text{length of the line segment connecting } P_i \text{ and } P_{i-1}) \end{aligned}$$

$$A_i = 2\pi \left(\frac{f(x_i) + f(x_{i+1})}{2} \right) \sqrt{1 + [f'(x_i^*)]^2} \cdot \Delta x$$

recall $f(x)$ is continuous and Δx is small

$$f(x_i^*) \approx f(x_i)$$

$$\therefore A_i \approx 2\pi f(x_i^*) \sqrt{1 + [f'(x_i^*)]^2} \cdot \Delta x$$

$$S \approx \sum_{i=1}^n 2\pi f(x_i^*) \sqrt{1 + [f'(x_i^*)]^2} \cdot \Delta x$$

$$\begin{aligned} S &= \lim_{n \rightarrow \infty} \sum_{i=1}^n 2\pi f(x_i^*) \sqrt{1 + [f'(x_i^*)]^2} \cdot \Delta x \\ &= \int_a^b 2\pi f(x) \sqrt{1 + [f'(x)]^2} \cdot dx \end{aligned}$$

$$2\pi \int_a^b f(x) \sqrt{1 + (f'(x))^2} dx,$$

as the derivative df/dx .[†] Since most of our surfaces of revolution is, we will instead use the formula

$$\phi_{\max} \int_a^b f(x) \sqrt{1 + (f'(x))^2} dx.$$

ace of revolution of a circular arc. The function that defines the profile is

$$f(z) = \sqrt{r^2 - z^2},$$

$$f'(z) = -\frac{z}{\sqrt{r^2 - z^2}}.$$

e is clipped at z_{\min} and z_{\max} . The surface area is therefore

$$\begin{aligned} A &= \phi_{\max} \int_{z_{\min}}^{z_{\max}} \sqrt{r^2 - z^2} \sqrt{1 + \frac{z^2}{r^2 - z^2}} dz \\ &= \phi_{\max} \int_{z_{\min}}^{z_{\max}} \sqrt{r^2 - z^2 + z^2} dz \\ &= \phi_{\max} \int_{z_{\min}}^{z_{\max}} r dz \\ &= \phi_{\max} r (z_{\max} - z_{\min}). \end{aligned}$$

confirming that the formula makes sense.

<<*Sphere Method Definitions*>>+= ▲▼

```
Float Sphere::Area() const {  
    return phiMax * radius * (zMax - zMin);  
}
```