Exercise 7

Download the project files *Laboratorio7Curves_Studenti*

• Implement the De Casteljeau algorithm for evaluating a Bezier curve starting from control vertices selected with the mouse on the viewport on the screen.

• Give the user the opportunity to view the control polygon.

• Experimentally verify that Bezier curves do not allow local control of the curve.

Download the project files Laboratorio7_Studenti

a) Implement the ability to select an object from the scene, by clicking with the left button on the object to be selected, as follows:

- Define an anchor point for each object in the scene.
- Trace the beam starting from the camera position in the direction identified with the click of the mouse.
- Calculate the intersection of the radius with a sphere centered at the anchor point of each object in the scene and predetermined radius. If there is an intersection of the ray with one of these objects, it identifies which object has been selected.

For tis aim, we need:

Write the function:

vec3 get_ray_from_mouse (float mouse_x, float mouse_y)

which takes as input the x, y coordinates of the mouse in the screen coordinate system, returns in vec3 the corresponding coordinates in the world coordinate system.

(remember to put z = -1 in the clipping space).

1) From screen coordinates to normalized coordinates (NDC)

#Mapping of x that belongs to the interval [A, B) in [l, r]

xm = (x-A) * (r-l) / (B-A) + l

Map xmouse from [0, width] to [-1,1]

Map ymouse from [heigth, 0] to [-1,1]

2) From normalized coordinates to clipping coordinates (z = -1),

3) Transform the clipping coordinates into View coordinates, pre-multiplying by the inverse of the Projection matrix.

4) Transform the view coordinates into world coordinates by pre-multiplying the view coordinates by the inverse of the View matrix.

Write the function

bool ray_sphere (vec3 ray_origin_wor, vec3 ray_direction_wor, vec3 sphere_centre_wor, float sphere_radius, float * intersection_distance)

* checks if a ray intersects a sphere. If not, it returns false. Rejects

the intersections behind the ray origin, and places intersection_distance at the closest intersection.

* /

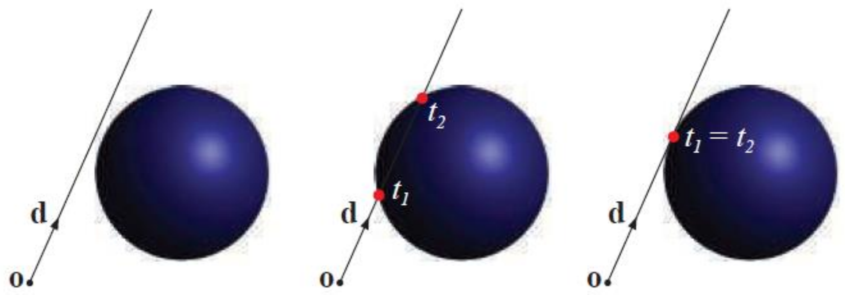


Figure 22.10. The left image shows a ray that misses a sphere and consequently $b^2 - c < 0$. The middle image shows a ray that intersects a sphere at two points ($b^2 - c > 0$) determined by the scalars t_1 and t_2. The right image illustrates the case where $b^2 - c = 0$, which means that the two intersection points coincide.

22.6.1 Mathematical Solution

A sphere can be defined by a center point, **c**, and a radius, r. A more compact implicit formula (compared to the one previously introduced) for the sphere is then

$$f(\mathbf{p}) = ||\mathbf{p} - \mathbf{c}|| - r = 0, \qquad (22.5)$$

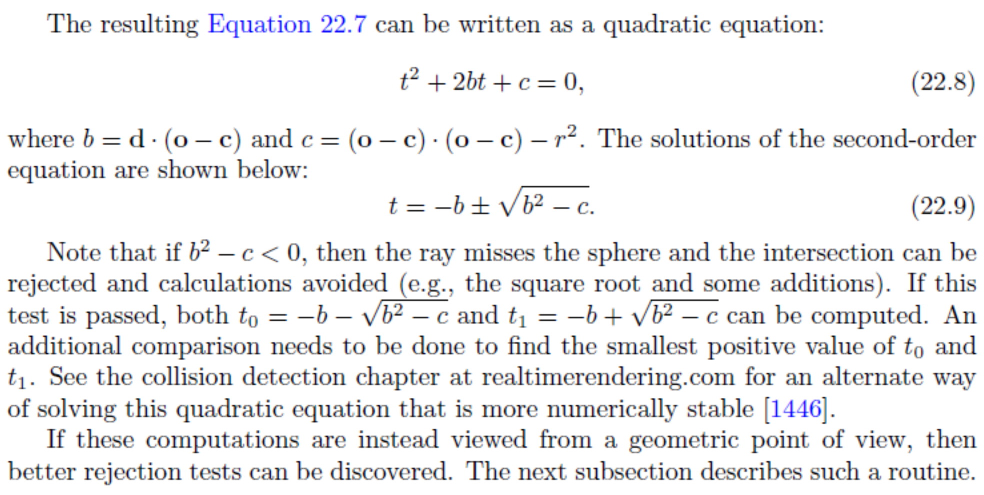
where **p** is any point on the sphere's surface. To solve for the intersections between a ray and a sphere, the ray $\mathbf{r}(t)$ simply replaces **p** in Equation 22.5 to yield

$$f(\mathbf{r}(t)) = ||\mathbf{r}(t) - \mathbf{c}|| - r = 0. \qquad (22.6)$$

Using Equation 22.1, that $\mathbf{r}(t) = \mathbf{o} + t\mathbf{d}$, Equation 22.6 is simplified as follows:

$$\begin{gathered} ||\mathbf{r}(t) - \mathbf{c}|| - r = 0 \\ \Longleftrightarrow \\ ||\mathbf{o} + t\mathbf{d} - \mathbf{c}|| = r \\ \Longleftrightarrow \\ (\mathbf{o} + t\mathbf{d} - \mathbf{c}) \cdot (\mathbf{o} + t\mathbf{d} - \mathbf{c}) = r^2 \qquad (22.7) \\ \Longleftrightarrow \\ t^2(\mathbf{d} \cdot \mathbf{d}) + 2t(\mathbf{d} \cdot (\mathbf{o} - \mathbf{c})) + (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - r^2 = 0 \\ \Longleftrightarrow \\ t^2 + 2t(\mathbf{d} \cdot (\mathbf{o} - \mathbf{c})) + (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - r^2 = 0. \end{gathered}$$

The last step comes from the fact that **d** is assumed to be normalized, i.e., $\mathbf{d} \cdot \mathbf{d} = ||\mathbf{d}||^2 = 1$. Not surprisingly, the resulting equation is a polynomial of the second order, which means that if the ray intersects the sphere, it does so at up to two points. See Figure 22.10. If the solutions to the equation are imaginary, then the ray misses the sphere. If not, the two solutions t_1 and t_2 can be inserted into the ray equation to compute the intersection points on the sphere.



The resulting Equation 22.7 can be written as a quadratic equation:

$$t^2 + 2bt + c = 0, \tag{22.8}$$

where $b = \mathbf{d} \cdot (\mathbf{o} - \mathbf{c})$ and $c = (\mathbf{o} - \mathbf{c}) \cdot (\mathbf{o} - \mathbf{c}) - r^2$. The solutions of the second-order equation are shown below:

$$t = -b \pm \sqrt{b^2 - c}. \tag{22.9}$$

Note that if $b^2 - c < 0$, then the ray misses the sphere and the intersection can be rejected and calculations avoided (e.g., the square root and some additions). If this test is passed, both $t_0 = -b - \sqrt{b^2 - c}$ and $t_1 = -b + \sqrt{b^2 - c}$ can be computed. An additional comparison needs to be done to find the smallest positive value of t_0 and t_1. See the collision detection chapter at realtimerendering.com for an alternate way of solving this quadratic equation that is more numerically stable [1446].

If these computations are instead viewed from a geometric point of view, then better rejection tests can be discovered. The next subsection describes such a routine.

Within the event mouse response function, pressing a button on the mouse Test for each object in the scene whether the ray originating in the camera and direction pointing to the mouse position intersects the sphere of predefined radius centered on the center of the object, and then take the object which is closest to observer.

b) Implement Camera moving forward (w key), backward (s key), left (a key), right (d key), along one direction.

c) Camera whose direction is defined by the passivemotion event.

d) Build a scene made up of 30 snowmen equally distributed in a 5x6 grid and visit the scene with the camera.