Ray Tracing

- Ray casting
  - Rays
  - Intersection tests
  - Intersection with scene graph
  - Lighting and material properties
- Recursive ray tracing
- Cheap tricks
- Optimizations

---

**Ray Tracing**

- Cast ray into scene
- Find intersection point (if any) that is closest to eye
- Compute luminance at intersection

---

**Find Intersection Point**

- Sphere
- Viewpoint

---

**Find Intersection Point**

- Intersection test hard!
- Squashed (aka transformed) sphere
- Viewpoint

---

**Find Intersection Point**

- Intersection test easy
  - Intersection test hard
- Sphere
- Viewpoint
- Object coordinates
- World coordinates
find intersection point

- world coordinates
- intersection point \( p \)
- \( R \)
- object coordinates
- intersection point \( p' \)
- \( M \)

does this make sense?

- is there an inverse transform \( M^{-1} \) for points?

Conceptually: scale

What operation inverts a scale by \( s \) in the \( x \)-direction?

For \( s \neq 0 \), scale by \( 1/s \) in the \( x \)-direction.

Any problem?

- We are not alone!

we are not alone...

the parallel universe view of homogenous coordinates

- we live in this universe
- it's not the only one, but it is the only one we can experience!
### Conceptually: rotate

What operation inverts a rotate by $\theta$ about the x-axis?

Rotate by $-\theta$ about the x-axis.

### Conceptually: translate

What operation inverts a translate by $dx$ in the x-direction?

Translate by $-dx$ in the x-direction.
translate

\[
\begin{pmatrix}
1 & 0 & 0 & -x_0 \\
0 & 1 & 0 & -y_0 \\
0 & 0 & 1 & -z_0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
1 & 0 & 0 & x_0 \\
0 & 1 & 0 & y_0 \\
0 & 0 & 1 & z_0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

does this make sense?

• is there an inverse transform \( M^{-1} \) for points? \( \text{YES!} \)

To invert composite \( M_3 \ldots M_1 \), use \( M_1^{-1} \ldots M_3^{-1} \).

• how do we apply a transform to a ray?

\( \text{YES!} \)

\( R = (p, v) \)

\( T(R) = (T(p), T(v)) \)

Does this make sense?

transforming vectors

\( p \)

\( v = p - q \) and \( T(v) = T(p) - T(q) \)

\( \text{rectangle shown for reference!} \)

How about this?

don’t worry about homogeneous coordinates until here

transforming vectors (tricky part)

\( p \)

\( q \)

\( v = p - q \) where \( q = (0, 0, 0, 1) \)

and \( T(v) = T(p-q) = T(p) - T(q) \)

\( \text{if translation involved then } T(q) = (0, 0, 0, 1) \)

find intersection point

\( p \)

\( q \)

\( T^{-1} \)

\( T^{-1} = T^{-1}(p) = T^{-1}(q) \)

\( T^{-1}(p) = M^{-1}p, T^{-1}(v) = (T^{-1}(M^{-1}(p-q))) / ||(M^{-1}(p-q))|| \)

\( \text{ray tracer skeleton has built-in call for this} \)

\( \text{intersection point } p' \)
**M and M⁻¹**

- Single transform:
  - Scale by \( s \)
  - Rotate by \( \theta \)
  - Translate by \( \Delta \)

- Composite transform:
  - \((M_3M_2M_1)^{-1}M_2^{-1}M_1^{-1}\)

**Scene graph traversal**

A sends the ray (represented relative to A’s coordinate system) to B.

- B converts the ray into its own coordinate system
  \( R_B = T(R_A) \)

- B computes the intersections of \( R_B \) with its objects
  \( \cdot B \) sends \( R_B \) to its children
  \( \cdot \) Each child returns its intersection info in B’s coordinate system
scene graph traversal

- B computes intersection closest to viewer, converts it to A's coordinate system and returns the info to A

Intersection info

- distance between intersection point and viewer
- point of intersection
- normal at point of intersection
- material properties of surface
- etc.

Intersection info

- distance between intersection point and viewer
- point of intersection
- normal at point of intersection
- material properties of surface
- etc.

Intersection info

- distance between intersection point and viewer
- point of intersection
- normal at point of intersection
- material properties of surface
- etc.

Intersection info

- distance between intersection point and viewer
- point of intersection
- normal at point of intersection
- material properties of surface
- etc.
The right way ...

N is normal to the tangent plane iff for any points p and q on the tangent plane \( N \cdot (p-q) = 0 \).

Assume N is normal to the tangent plane and \( QN \) is normal to the tangent plane transformed by \( M \).

\( Q \) must satisfy the following for any points p and q on the tangent plane:
\[
N^T(p-q) = 0 \iff (QN)^T(M(p-q)) = 0
\]
\[
N^T(p-q) = 0 \iff N^T(QM)(p-q) = 0
\]
Thus \( Q = (M^{-1})^T \)^T

ray casting

- cast ray through pixel into scene
- find intersection point (if any) that is closest to eye
- compute color at intersection