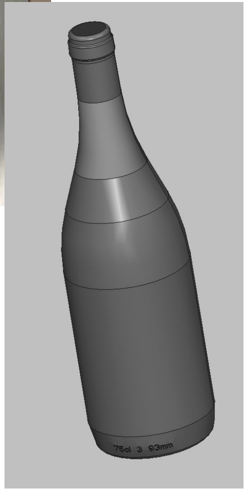
Glass Container Modeling and Rendering given a Glass Mold

Caelan Booker
Joe Fradley

March 8th, 2021

Glass Mold 3D Model

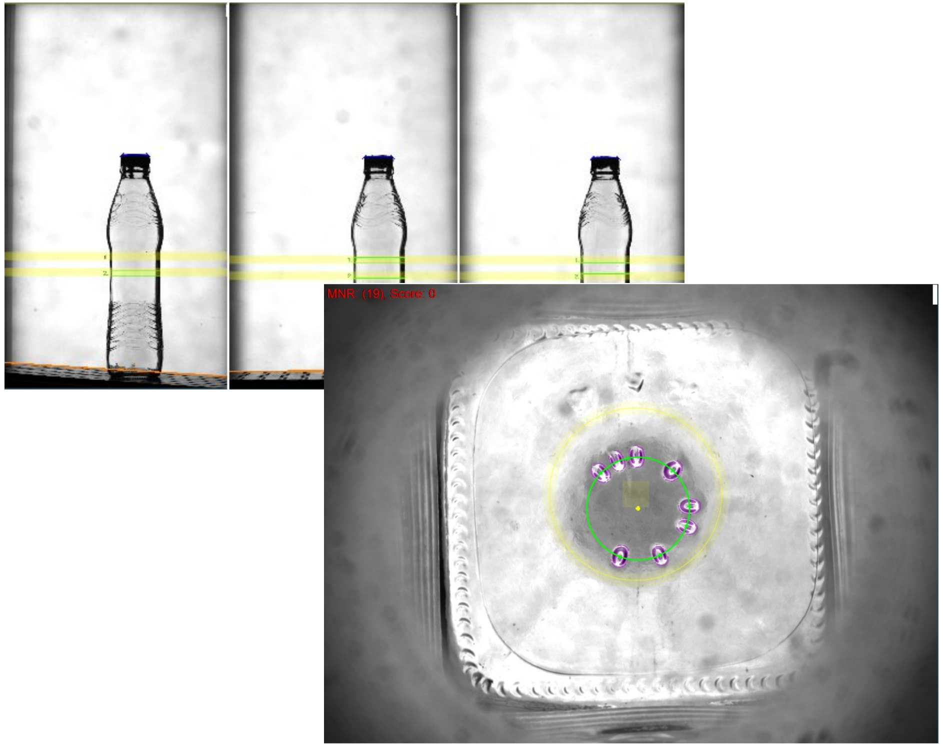
- Mold forms a cavity where molten glass is placed and then filled with air to form a container
- Typically CAD software is used to design the outer shell of the container, which is used to manufacture the mold out of metal
- The 3D model is solid
- Standard CAD file format, such as STEP (ISO 10303-21)
- The resulting container has a number of features not present in the 3D model which include:
  - Hollow with variable wall thickness
  - Transparent with variable refractive index
  - Different color glass
  - Potential defects due to manufacturing variables



Rendered with FreeCAD

Motivation

- Give designers the ability to view rendering of the final container
- Assist inspection equipment by generating a “golden image” of what the given container should look like at various stations

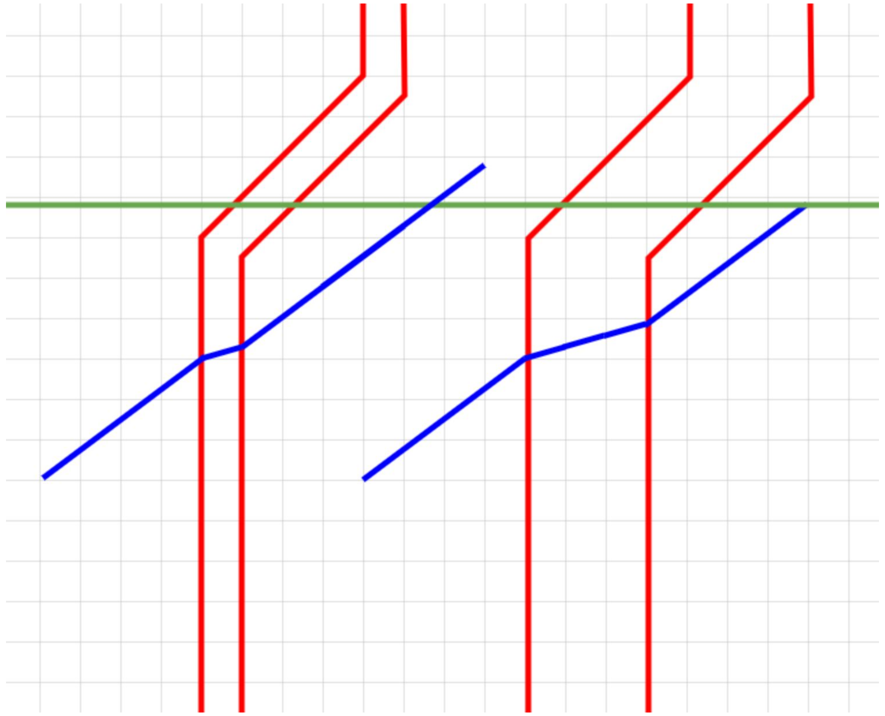


Hollow Meshes

- Actual bottles are hollow and wall thickness can vary. These properties, along with the material being used for the bottle, affect the way bottle interacts with light.
- Just changing the thickness of the bottle alone alters how light bends significantly.
- Naive solution:

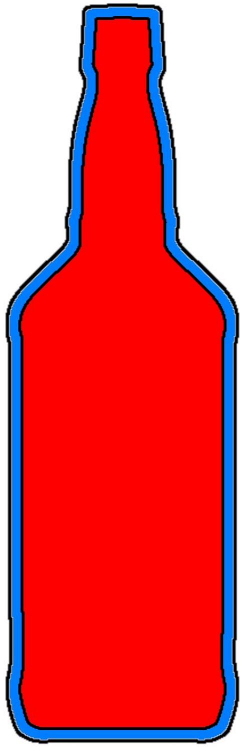
```
When ray-bottle intersection occurs:
    Change ray direction based on material
    Travel fixed distance based on wall thickness
    Change direction back and continue travelling
```

- Doesn't account for varying wall thickness depending on where ray intercepts, and doesn't account for the possibility of the ray hitting the other side of the bottle!



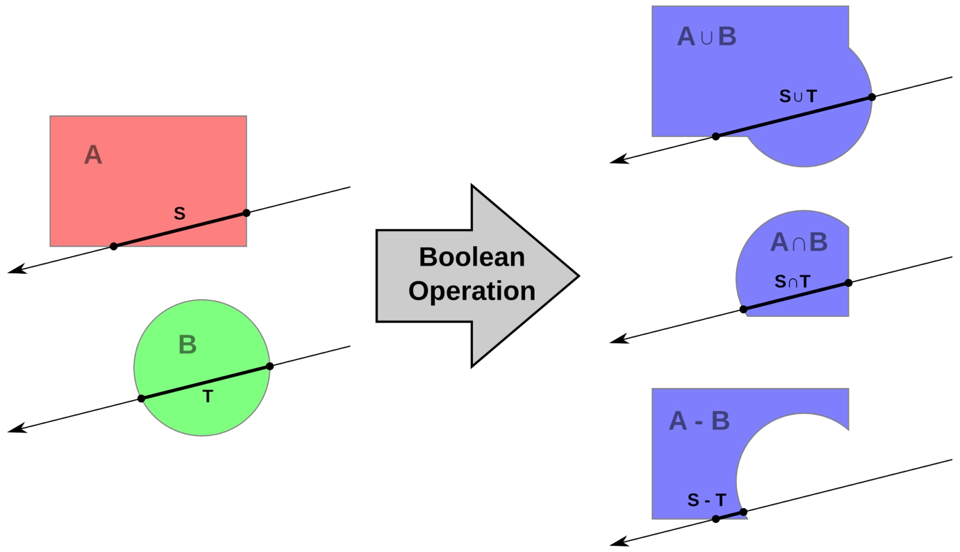
Other Approaches to Hollowing

- Constructive Geometry
  - Not directly responsible for hollowing, but can be paired with the options below to represent the hollowed model in the world.
- Simple but inaccurate: Scale the original model
- Very complex: Use vertex data of original mesh to construct an inner mesh
- In-between: Use a surface of revolution purpose built for the profile of the mesh
- Other options: Just make the actual model itself out of a surface of revolution



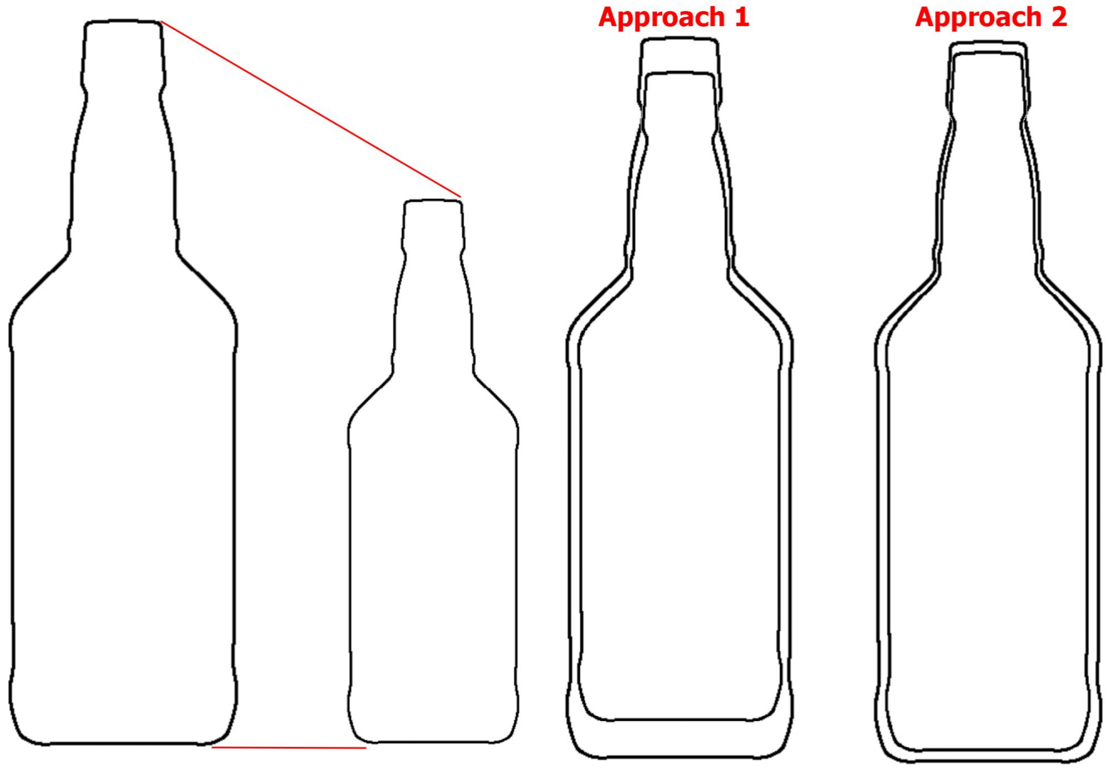
Constructive Geometry

- Instead of modifying the original mesh to contain the wall data, we can also just use constructive geometry and two separate objects.
    - They don't need to share the same type! Can do mesh + surface of revolution.
- Ray trace through the first object and find the intervals in which the ray is inside the object. Then do the second object and find the intervals for when the ray is in that object.
- From there, modify the intervals according to the operation we want to perform.



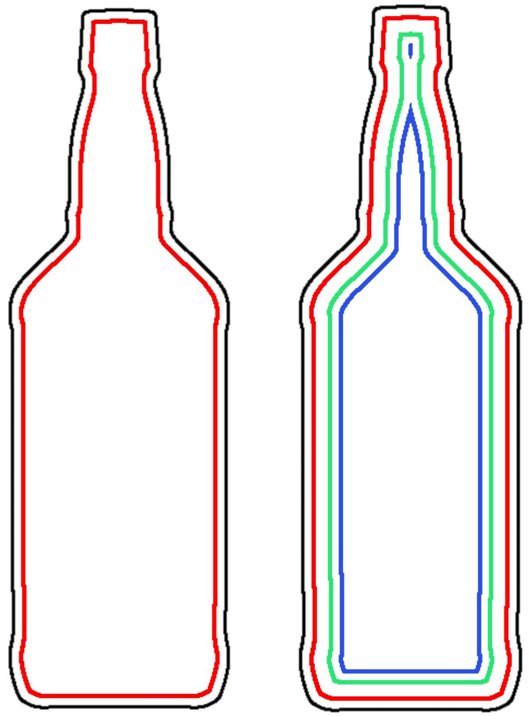
Simple Method: Scaling

- Make a scaled down copy of the original mesh and center it inside the full sized version.
- Approach 1: Scale the entire mesh by the same value, and center
  - Problem: Certain dimensions will be more affected than others
- Approach 2: Individually scale certain dimensions
  - Still not perfect



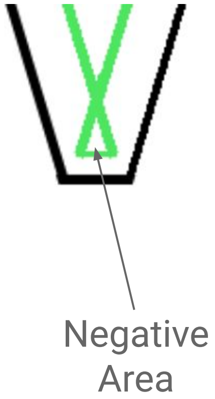
Complex Method: Building an Inner Mesh

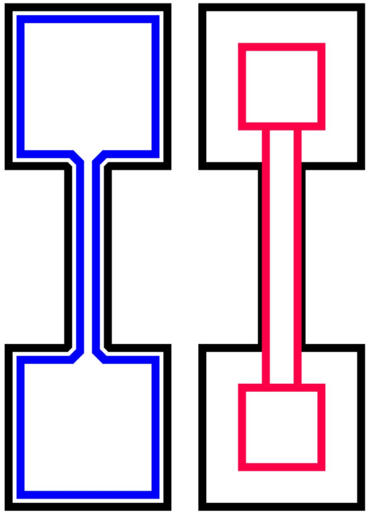
- It’s possible to use the vertex data to construct a mesh representing the inside walls of the structure.
- Move the vertices inwards and resolve conflicts to create new inner triangles.
- Not quite that simple, lots of edge cases.
- The big 2 edge cases we noticed:
    - How do we drop vertices when they aren’t needed?
    - How do we determine when one inside wall has overlapped with another?



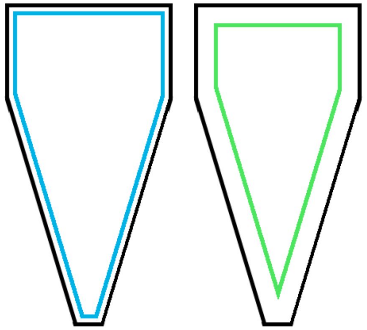
Edge Cases

- These two edge cases are closely linked, but different enough to warrant two examples. Both of these conditions arise due to wall thickness.
- If the wall thickness is large enough, there are situations where the adjacent faces of a surface might overlap and cut that surface out of the final result
- There can also be situations where non-adjacent faces overlap.
- Both of these create “negative areas“ that should be solid instead of hollow.
- Still, this approach should be possible, but **very** math intensive.





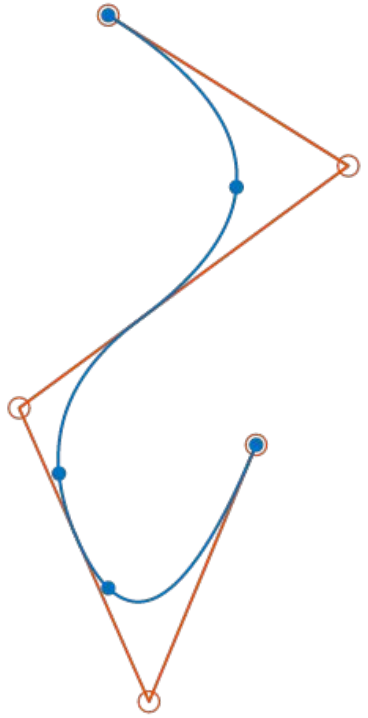
The middle walls on the right diagram have overlapped. This means there should not be any hollow area in the thin section for the desired wall thickness.



On the left diagram, we can see all of the faces of the original mesh are mirrored in the hollowed inner mesh. On the right however, the wall thickness is large enough that the bottom face is dropped from the inner mesh.

Surfaces of Revolution

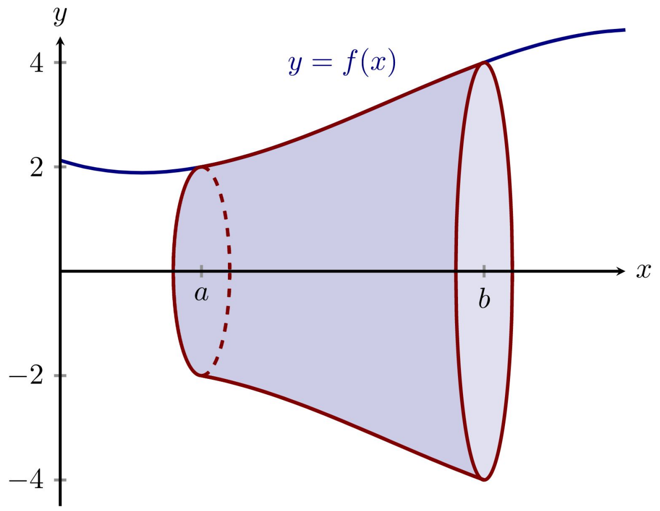
- Curves rotated around an axis in 3D space.
- Technically, the curve can be just about any function.
- But, we want to use Bezier curves! This gives us a lot of control over the shape, and is fairly intuitive to use.
- But we did this in 2D. Can we do Bezier curves in 3D?
- Yes: Just need a new set of parameters for the z dimension.
  - Before, we would create two 4-vectors of all the x and y values of our 2D control points to generate two sets of coefficients, one for producing the x values of the curve and one for the y values.
  - Just do this one more time with the z values of the control points.



Surfaces of Revolution (cont.)

- A Surface of Revolution is defined by 3 things:
    - A function representing the curve
        - In our case, 4 points to make a bezier curve with
    - An axis to rotate around
    - A base point for that axis
- From this, we can construct the coefficients we need for the function f(u), and ray trace.
- Except, the ray tracing part needs **a lot** of math.
  Let’s take a look at all that math!

A basic surface of revolution, not using bezier curves.



Surfaces of Revolution (cont.)

- Let’s start by making the matrix A that will produce points along the curve given a value ***u*** between 0 and 1.

$$f(u) = a_0 1 + a_1 u + a_2 u^2 + a_3 u^3$$

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{3} & 1 & 0 & 0 \\ 1 & \frac{2}{3} & \frac{1}{3} & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

$$B = C^{-1}$$

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix}$$

$$\vec{a_x} = \begin{bmatrix} a_{x0} \\ a_{x1} \\ a_{x2} \\ a_{x3} \end{bmatrix} = B * \begin{bmatrix} p_{x0} \\ p_{x1} \\ p_{x2} \\ p_{x3} \end{bmatrix}$$

x-coordinate of point on curve at u $= \vec{a_x} \bullet \begin{bmatrix} 1 \\ u \\ u^2 \\ u^3 \end{bmatrix}$

All 3 coordinates for point at position u $= \begin{bmatrix} \vec{a_x} \\ \vec{a_y} \\ \vec{a_z} \end{bmatrix} \begin{bmatrix} 1 \\ u \\ u^2 \\ u^3 \end{bmatrix}$

Same thing as above, $\begin{bmatrix} a_{x0} & a_{x1} & a_{x2} & a_{x3} \\ a_{y0} & a_{y1} & a_{y2} & a_{y3} \\ a_{z0} & a_{z1} & a_{z2} & a_{z3} \end{bmatrix} \begin{bmatrix} 1 \\ u \\ u^2 \\ u^3 \end{bmatrix}$

Surfaces of Revolution (cont.)

- Then let's make our future work easier by doing a translate and change of basis on the result of f(u) such that our base point is the origin and we are rotating around the y axis.
- By doing this, all we really need to do before ray tracing is translating the origin of our ray, and then performing a change of basis on the ray origin and direction. Then we can raytrace like normal.

Matrix T to translate from base point b to origin $= \begin{bmatrix} 1 & 0 & 0 & -b_x \\ 0 & 1 & 0 & -b_y \\ 0 & 0 & 1 & -b_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Basis vector $\vec{v} = normalize(axis)$

Make a copy of $\vec{v}$ called $\vec{t}$ and set it's smallest magnitude element to 1.

Basis vector $\vec{u} = normalize(\vec{t} \times \vec{v})$

Basis vector $\vec{w} = \vec{v} \times \vec{u}$

Matrix Q to do a change of basis with basis vectors $\vec{u}, \vec{v}, \vec{w} = \begin{bmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1}$

Want Q * T * A, but Q and T are affine transformations, and A produces a 3-vector.

Hack. A $= \begin{bmatrix} a_{x0} & a_{x1} & a_{x2} & a_{x3} \\ a_{y0} & a_{y1} & a_{y2} & a_{y3} \\ a_{z0} & a_{z1} & a_{z2} & a_{z3} \\ 1 & 0 & 0 & 0 \end{bmatrix}$

Solving

$$\begin{bmatrix} o_x + td_x \\ o_y + td_y \\ o_z + td_z \end{bmatrix} = \begin{bmatrix} cosP & 0 & sinP \\ 0 & 1 & 0 \\ -sinP & 0 & cosP \end{bmatrix} \begin{bmatrix} f_x(u) \\ f_y(u) \\ f_z(u) \end{bmatrix}$$

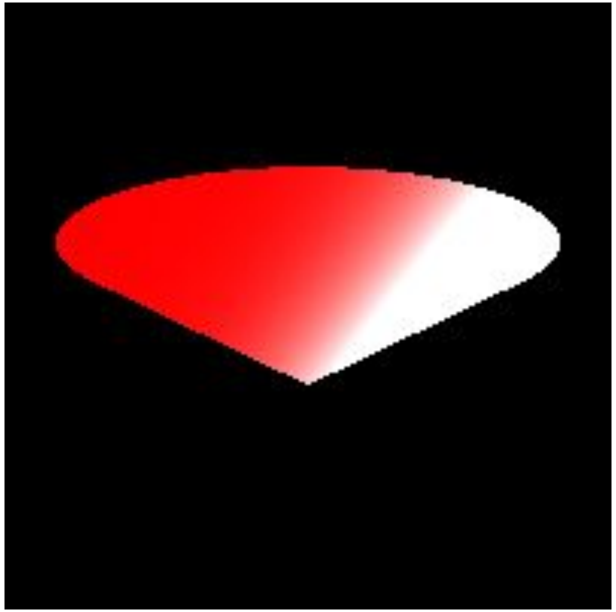
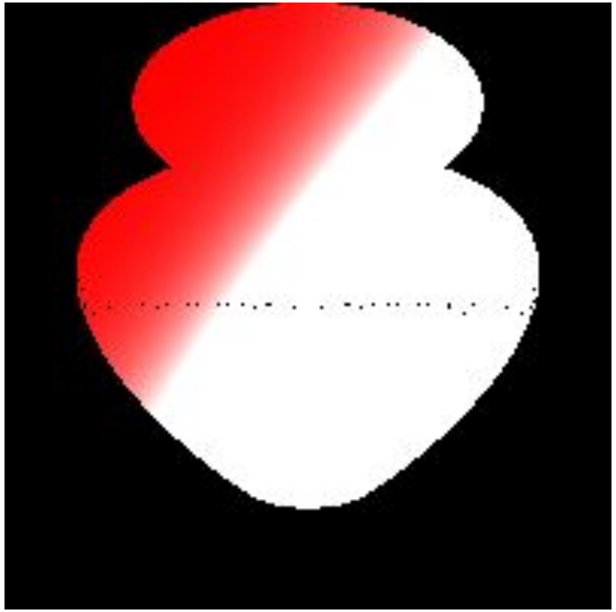
$$o_y + td_y = f_y(u) \rightarrow t = \frac{f_y(u) - o_y}{d_y}$$

$$f_x^2(u) + f_z^2(u) = (o_x + td_x)^2 + (o_z + td_z)^2$$

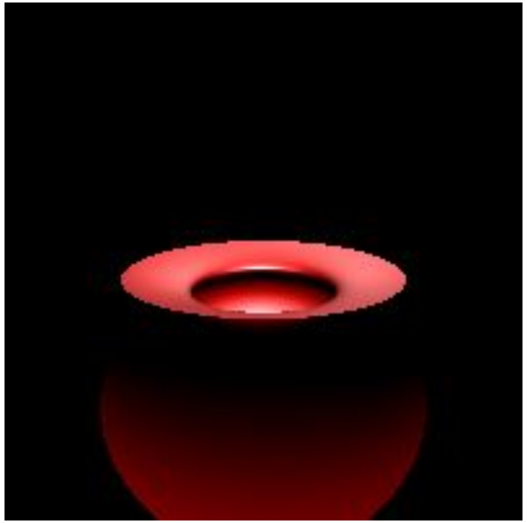
$$tan(P) = \frac{o_z + td_z}{o_x + td_x}$$

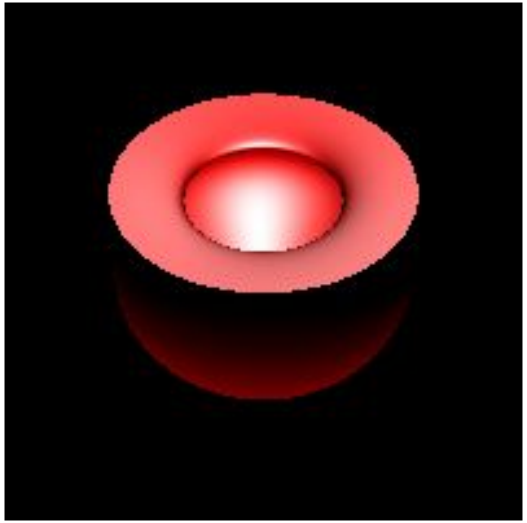
Normal at an intersect is the cross product of the derivatives in the u and P directions.

Results!

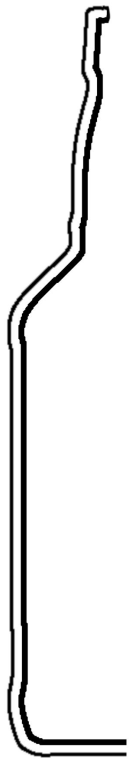
Nicer Results!





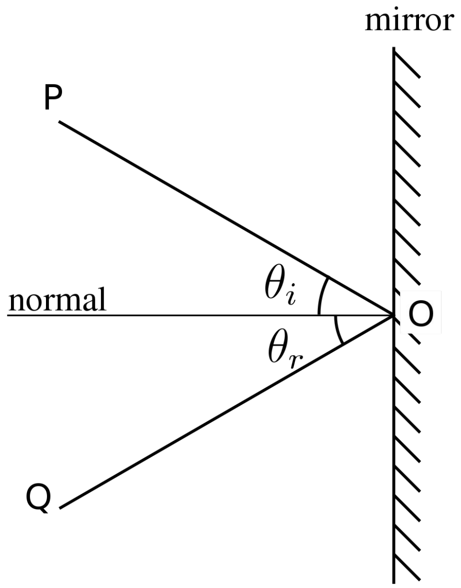
Uses for Hollowing

- A surface of revolution can be used to hollow out the inside of a mesh. Just need to choose some curves that represent the shape we want. Can use the outside of the bottle for a good starting point.
- Can also just model the bottle itself using a surface of revolution! Or other things, such as a wine glass.



Reflection (Recap)

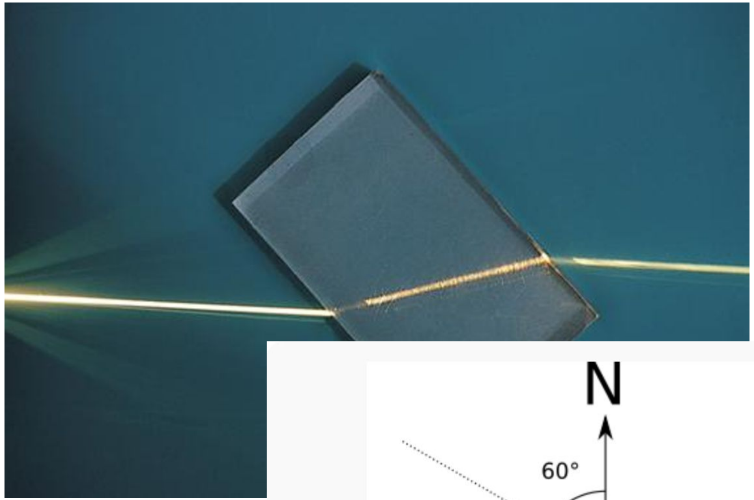
- The reflected angle is equal to the angle of incidence with respect to the normal vector
- $\theta_i == \theta_r$
- Once θ_r is computed, continue tracing the ray from the intersection point along that new direction
- Assignment “a2” used a “mirror coefficient” to compute what percentage of the color came from the object color and that determined by the reflected ray:
    - `final_color = (mirror_coeff * reflected_color) + (1 - mirror_coeff) * local_color`



Refraction (aka Transmission)

- Light changes direction when it travels from one medium to another with a different refractive index
- This new direction can be computed with Snell's law, where α_1 and n_1 are the angle of incidence and refractive index of the first medium, and α_2 and n_2 of the second.

$$\frac{\sin(\alpha_1)}{\sin(\alpha_2)} = \frac{n_2}{n_1}$$



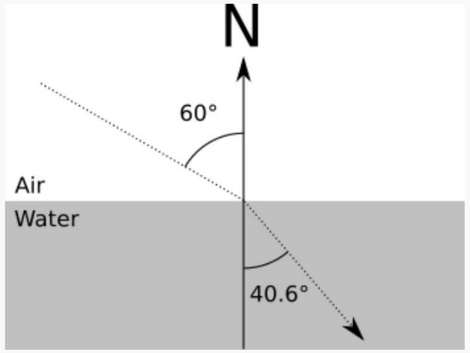
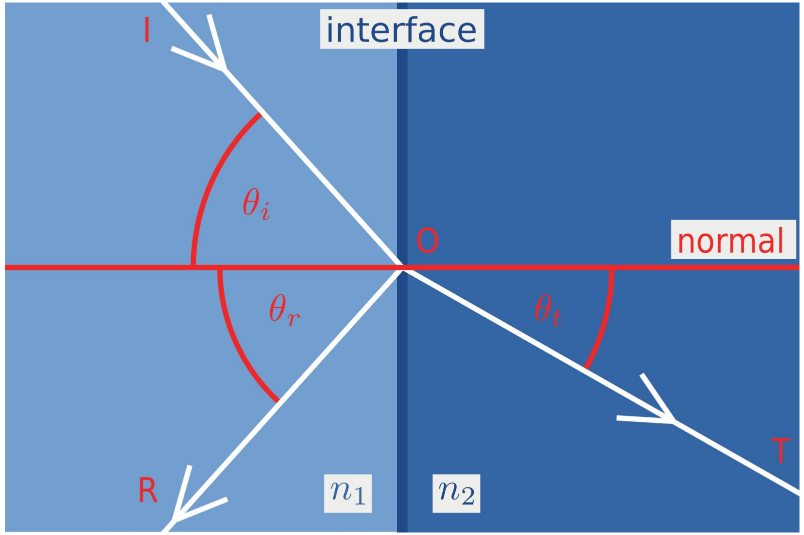


Figure 5-5: A ray of light is refracted (changes direction) as it leaves air and enters water.

Reflection + Refraction Ratio: Fresnel Equations

- Transparent objects can both reflect and transmit light
- The amount of light reflected can be determined using Fresnel Equations
- Using conservation of energy we can compute the amount of light transmitted by subtracting what was reflected:
    - $R_t = 1 - R_r$



Fresnel Equations

- Light is made up of two waves perpendicular to each other, referred to as parallel and perpendicular polarised light
- The amount of reflected light for each of these components(R_p = parallel, R_s = perpendicular) can be computed with the following formulas:

$$R_\mathrm{s} = \left| \frac{Z_2 \cos\theta_\mathrm{i} - Z_1 \cos\theta_\mathrm{t}}{Z_2 \cos\theta_\mathrm{i} + Z_1 \cos\theta_\mathrm{t}} \right|^2,$$

$$R_\mathrm{p} = \left| \frac{Z_2 \cos\theta_\mathrm{t} - Z_1 \cos\theta_\mathrm{i}}{Z_2 \cos\theta_\mathrm{t} + Z_1 \cos\theta_\mathrm{i}} \right|^2,$$

- For most "natural light" applications, taking the average of these will give the overall effective percent of light reflected:

$$R_\mathrm{eff} = \frac{1}{2}\left(R_\mathrm{s} + R_\mathrm{p}\right)$$

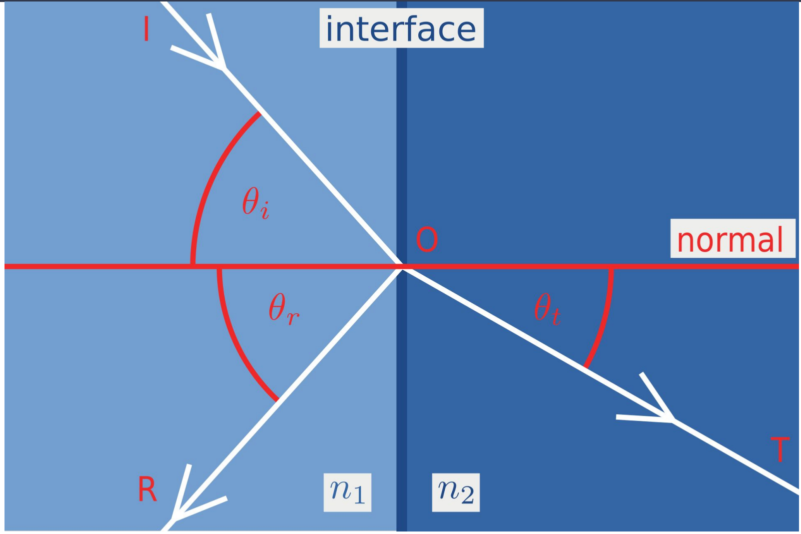
https://en.wikipedia.org/wiki/Fresnel_equations

Fresnel Equations

- Therefore the percentage of light refracted is:

$$T_{\text{eff}} = 1 - R_{\text{eff}}$$

- Implementation:
  - Trace a ray along the reflected ray
  - Trace a ray along the refracted ray

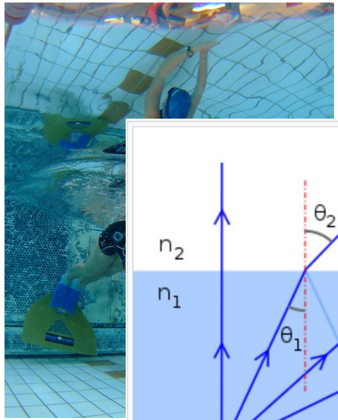


```
final_color = local_color + (reflected_color * Fr) + refracted_color * (1 - Fr)
```

Total internal reflection

- Can occur when light travels to a medium with a lower refractive index, such as water to air.
- If the refracted angle is greater or equal to the “critical angle” then total internal reflection occurs
- Equation for the critical angle given to refractive index values:

$$\theta_c = \arcsin(n_2/n_1)$$



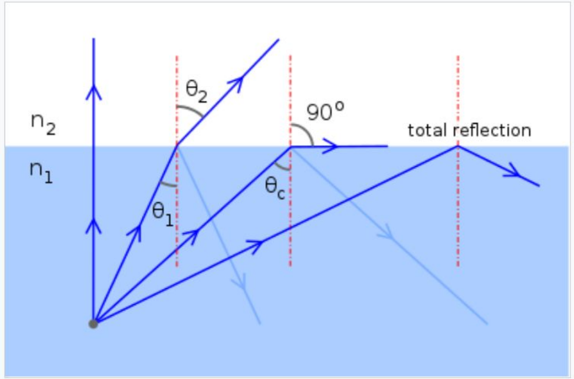
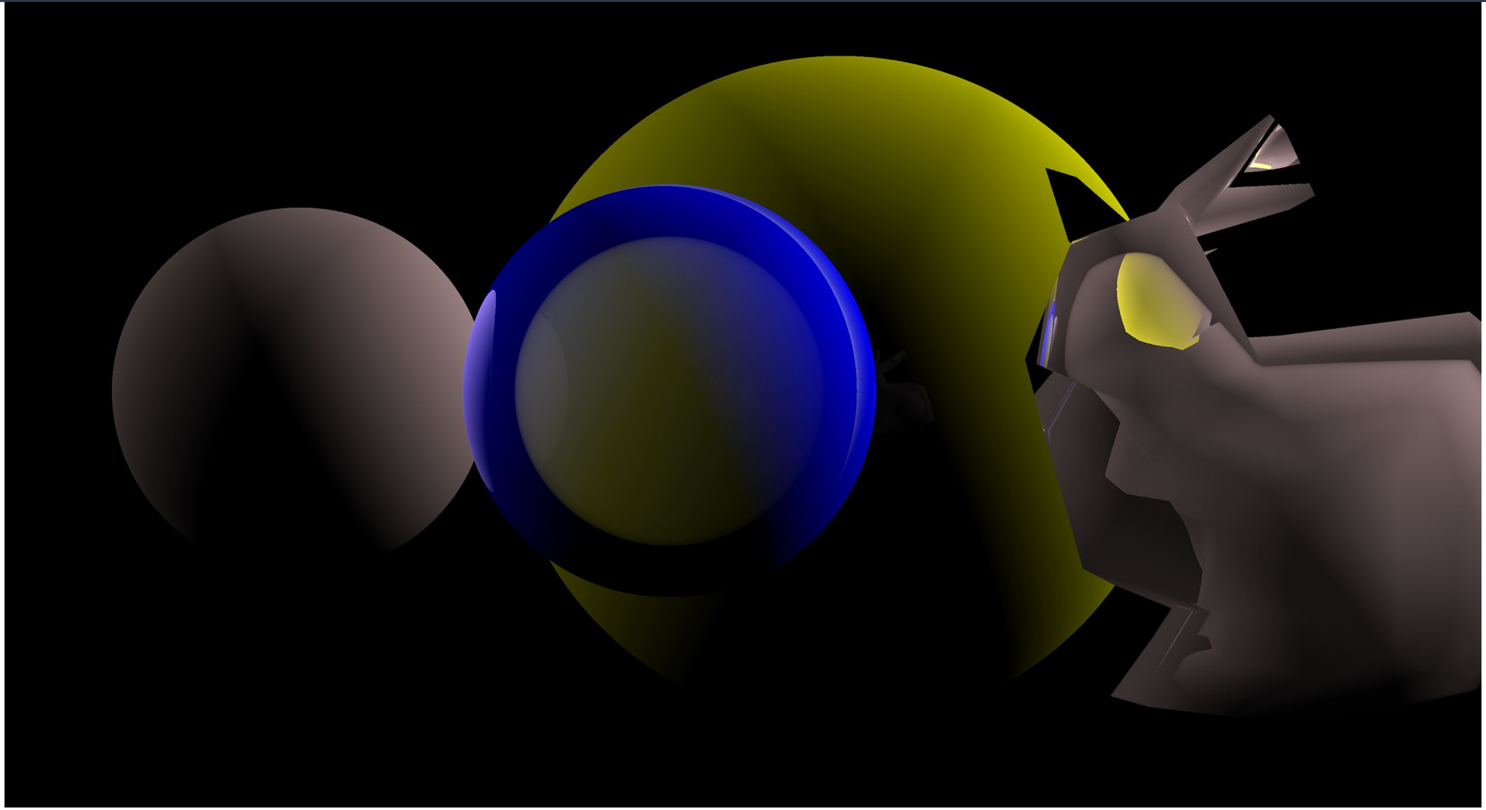


Fig. 5: Behavior of a ray incident from a medium of higher refractive index n_1 to a medium of lower refractive index n_2, at increasing angles of incidence.[Note 2]

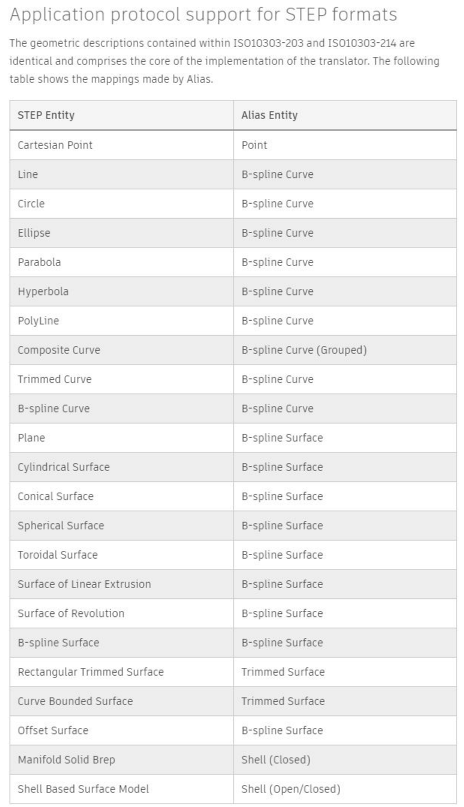
https://en.wikipedia.org/wiki/Total_internal_reflection

Results



STEP File format

- STEP: "STandard for the Exchange of Product model data"
- ISO 10303
- Supports more complex surfaces then just simple mesh support of OBJ files
- Plain text



Application protocol support for STEP formats

The geometric descriptions contained within ISO10303-203 and ISO10303-214 are identical and comprises the core of the implementation of the translator. The following table shows the mappings made by Alias.

STEP Entity	Alias Entity
Cartesian Point	Point
Line	B-spline Curve
Circle	B-spline Curve
Ellipse	B-spline Curve
Parabola	B-spline Curve
Hyperbola	B-spline Curve
PolyLine	B-spline Curve
Composite Curve	B-spline Curve (Grouped)
Trimmed Curve	B-spline Curve
B-spline Curve	B-spline Curve
Plane	B-spline Surface
Cylindrical Surface	B-spline Surface
Conical Surface	B-spline Surface
Spherical Surface	B-spline Surface
Toroidal Surface	B-spline Surface
Surface of Linear Extrusion	B-spline Surface
Surface of Revolution	B-spline Surface
B-spline Surface	B-spline Surface
Rectangular Trimmed Surface	Trimmed Surface
Curve Bounded Surface	Trimmed Surface
Offset Surface	B-spline Surface
Manifold Solid Brep	Shell (Closed)
Shell Based Surface Model	Shell (Open/Closed)

https://knowledge.autodesk.com/support/alias-products/learn-explore/caas/CloudHelp/cloudhelp/2019/ENU/Alias-ImportingExporting/files/GUID-A6F13C9D-9A69-4B72-A25C-55A144B8535F-htm.html

STEP File Sample

```
#7145=CARTESIAN_POINT('',(1.956590335226E-14,1.489368855047E2,
-3.822341141259E1));
#7146=CARTESIAN_POINT('',(2.481786542645E-14,1.341581379339E2,
-4.090473771141E1));
#7147=CARTESIAN_POINT('',(2.096156146846E-14,1.191385086726E2,
-4.079690759637E1));
#7148=CARTESIAN_POINT('',(2.021343062411E1,1.629613349067E2,-3.284607848130E1));
#7149=CARTESIAN_POINT('',(2.352263376723E1,1.489368855047E2,-3.822341141259E1));
#7150=CARTESIAN_POINT('',(2.517271821043E1,1.341581379339E2,-4.090473771141E1));
#7151=CARTESIAN_POINT('',(2.510635970889E1,1.191385086726E2,-4.079690759637E1));
#7152=CARTESIAN_POINT('',(3.405047605874E1,1.629613349067E2,-1.811112170376E1));
#7153=CARTESIAN_POINT('',(3.962498463641E1,1.489368855047E2,-2.107614936195E1));
#7154=CARTESIAN_POINT('',(4.240462960973E1,1.341581379339E2,-2.255461586909E1));
#7155=CARTESIAN_POINT('',(4.229284558801E1,1.191385086726E2,-2.249515901984E1));
#7156=CARTESIAN_POINT('',(3.278126424839E1,1.629613349067E2,2.062422332341E0));
#7157=CARTESIAN_POINT('',(3.814798624147E1,1.489368855047E2,2.400067860779E0));
#7158=CARTESIAN_POINT('',(4.082402155533E1,1.341581379339E2,2.568429732110E0));
#7159=CARTESIAN_POINT('',(4.071640421840E1,1.191385086726E2,2.561659023166E0));
#7160=(BOUNDED_SURFACE()B_SPLINE_SURFACE(3,3,((#7132,#7133,#7134,#7135),(#7136,
#7137,#7138,#7139),(#7140,#7141,#7142,#7143),(#7144,#7145,#7146,#7147),(#7148,
#7149,#7150,#7151),(#7152,#7153,#7154,#7155),(#7156,#7157,#7158,#7159)),
.UNSPECIFIED.,.F.,.F.,.F.)B_SPLINE_SURFACE_WITH_KNOTS((4,3,4),(4,4),(0.E0,1.E0,
2.E0),(0.E0,1.E0),.UNSPECIFIED.)GEOMETRIC_REPRESENTATION_ITEM()RATIONAL_B_SPLINE_SURFACE(((1.197664505200E0,1.183795474926E0,1.183795474926E0,1.197664505200E0),(
9.457933490053E-1,9.348410025564E-1,9.348410025564E-1,9.457933490053E-1),(
9.457933490053E-1,9.348410025564E-1,9.348410025564E-1,9.457933490053E-1),(
1.197664505200E0,1.183795474926E0,1.183795474926E0,1.197664505200E0),(
9.457933490053E-1,9.348410025564E-1,9.348410025564E-1,9.457933490053E-1),(
9.457933490053E-1,9.348410025564E-1,9.348410025564E-1,9.457933490053E-1),(
1.197664505200E0,1.183795474926E0,1.183795474926E0,1.197664505200E0)))REPRESENTATION_ITEM('')SURFACE());
```