Well & Pipette Tip Geometry

Robert Atkinson
18 November 2019

This Mathematica notebook explores the geometry and shape of various wells, pipettes, and pipette tips. The goal is to create simple, closed-form functions for such things as the depth of liquid as a function of liquid volume, and the amount of radial clearance available when a given pipette tip is mounted on a given pipette and is inserted in a will at a certain depth. Applications of these functions include the ability to know where the top of liquid is in any given tube, facilitating the best-practice of pipetting at the top of liquid rather than the bottom, and the ability to ‘touch-tip’ at any depth in a well, not just the top, which in turn can lead to better mixing logic.

Programmatic Utilities

This section contains several utilities needed in the sequel.

```
(*FrontEndExecute[{FrontEndToken[InputNotebook[],"SelectAll"]}];
FrontEndExecute[{FrontEndToken[InputNotebook[],"SelectionOpenAllGroups"]}];*)
```

```
On[Assert]
Clear[assert]
assert[expr_] := assert[expr, "assertion failed"]
assert[expr_, msg_] := Module[{value = Evaluate[expr]},
  If[BooleanQ[value],
   Assert[value, Row[{msg, ": ", HoldForm[expr]}]]
   ,
   Assert[value, Row[{msg, ": ", HoldForm[expr]}]] (* to do: improve message *)
  ]
 ]
SetAttributes[assert, HoldAll]
assert[False]
assert[3]
```

... Assert: Assertion value$1837 in Assert[value$1837, assertion failed: False] failed.

... Assert: Assertion test value$1857 evaluated to 3 that is neither True nor False.

... Assert: Assertion value$1857 in Assert[value$1857, assertion failed: 3] failed.

```
printCell[cell_] := CellPrint[ExpressionCell[cell, "Output"]]
cellPrint[cell_] := CellPrint[ExpressionCell[cell, "Output"]]
```

```
log[msg_] := CellPrint[TextCell[msg, "Text"]]
```

```
test[expr_] := Module[{evald},
  evald = Evaluate[expr];
  printCell[HoldForm[expr] → evald];
  evald]
test2[expr_] := Module[{evald},
  printCell[HoldForm[expr] → "evaluating..."];
  evald = Evaluate[expr];
  printCell["..." → evald];
  evald]
SetAttributes[test, HoldAll]
SetAttributes[test2, HoldAll]
```

```
complement[angle_] := π / 2 - angle
```

```
Clear[hasImaginary]
hasImaginary[expr_] := Module[{result},
  (*result = Reap[Scan[Function[ee, If[ee ≠ Conjugate[ee], Sow[True]]],{expr}, {-1, Infinity}]];*)
  result = Scan[Function[ee, If[ee ≠ Conjugate[ee], Return[True]]], {expr}, {-1, Infinity}];
  (*Length @ result[[2]] > 0 *)
  result === True]
SetAttributes[hasImaginary, HoldAll]
test @ hasImaginary[1 + 2 I];
test @ hasImaginary[30 !];
```

```
hasImaginary[1 + 2 i] → True
```

```
hasImaginary[30 !] → False
```

```
toDeg[rad_] := rad / Pi * 180
toRadian[deg_] := deg / 180 * Pi
```

```
Clear[variables, unboundQ]

unboundQ[x_Symbol] := True
unboundQ[_] := False
unboundQ[E] := False
unboundQ[I] := False
unboundQ[Pi] := False
unboundQ[π] := False

variables[expr_] := variables[expr, {}]
variables[expr_, except_] := Module[{result, reaped},
   {result, reaped} = Reap[Scan[(If[unboundQ[#], Sow[#]]) &, expr, Infinity]];
   If[Length[reaped] == 0,
    {}
    ,
    Complement[reaped[[1]] // Union, except]
   ]];

test @ variables[e == m c^2];
test @ variables[{C[1]}];
```

```
variables[e == m c^2] → {c, e, m}
```

```
variables[{c1}] → {}
```

```
Clear[genericize]

genericize[expr_] := genericize[expr, NumberQ, {} ]
genericize[expr_, test_] := genericize[expr, test, {}]
genericize[expr_, test_, except_] := Module[{result, reaped, numbers, count, neg, pos, zero, constants, constraints, rules},
   {result, reaped} = Reap[Scan[ (If[test[#], Sow[#]]) &, expr, Infinity]];
   If[Length[reaped] > 0,
    numbers = reaped[[1]] // Union; (* todo: should we merge duplicates like this? It does make the logic below somewhat easier ... *)
    numbers = Complement[numbers, except];
    numbers = Sort[numbers];
    count = Length[numbers];
    constants = C[#] & /@ Range[count];
    If[count > 1,
     constraints = constants[[#]] < constants[[# + 1]] & /@ Range[count - 1]
     ,
     constraints = {}
    ];
    pos = Select[numbers, # > 0 &];
    neg  = Select[numbers, # < 0 &];
    zero = Select[numbers, # == 0 &];
    If[Length[pos] > 0, constraints = Append[constraints, constants[[count - Length[pos] + 1]] > 0]];
    If[Length[neg] > 0, constraints = Append[constraints, constants[[Length[neg]]] < 0]];
    If[Length[zero] > 0, constraints = Append[constraints, constants[[Length[neg] + 1]] == 0]];
    constraints = And @@ constraints;

    rules = (numbers[[#]] → constants[[#]]) & /@ Range[count]
   ];
   {expr /. rules, Reverse[rules, {2}], constraints}
  ];

test @ genericize[cone[h, 2] + fred[1.2, 3.14159, 1.2, seven, -2, -3, 0]];
```

```
genericize[cone[h, 2] + fred[1.2, 3.14159, 1.2, seven, -2, -3, 0]] → {cone[h, ℭ5] + fred[ℭ4, ℭ6, ℭ4, seven, ℭ2, ℭ1, ℭ3],
  {ℭ1 → -3, ℭ2 → -2, ℭ3 → 0, ℭ4 → 1.2, ℭ5 → 2, ℭ6 → 3.14159}, ℭ1 < ℭ2 && ℭ2 < ℭ3 && ℭ3 < ℭ4 && ℭ4 < ℭ5 && ℭ5 < ℭ6 && ℭ4 > 0 && ℭ2 < 0 && ℭ3 == 0}
```

```
Clear[enumerate]
enumerate[iterable_] := MapThread[{#1, #2} &, {Range[Length[iterable]], iterable}]
enumerate[func_, iterable_] := MapThread[func[#1, #2] &, {Range[Length[iterable]], iterable}]

test @ enumerate[{a, b, c}];
Function[{i, value}, value + i] @@ # & /@ enumerate[{a, b, c}]
```

```
enumerate[{a, b, c}] → {{1, a}, {2, b}, {3, c}}
```

```
{1 + a, 2 + b, 3 + c}
```

```
Clear[pairUp]
pairUp[a_, b_] := Transpose[{a, b}]
pairUp[a_, b_, c_] := Transpose[{a, b, c}]
pairUp[{1, 2, 3}, {a, b, c}, {do, re, mi}]
```

```
{{1, a, do}, {2, b, re}, {3, c, mi}}
```

```
Clear[qReduce]
qReduce[expr_, vars_, dom_ : Reals] := Quiet[Reduce[expr, vars, dom], {Reduce::ratnz}]
```

Geometric Shapes

This section contains definitions of the volume, height, etc of several different mathematical shapes

Utilities

```
volumeFromDepthUsingInverse[shape_, depth_] := InverseFunction[Function[v, depthFromVolume[shape, v]]][depth]
```

```
Clear[genericVolumeFromDepthUsingInverse]
genericVolumeFromDepthUsingInverse[genericShape_, depth_] := Module[{result},
  result = FullSimplify[volumeFromDepthUsingInverse[genericShape, depth], assumptions[genericShape]];
  genericVolumeFromDepthUsingInverse[genericShape, depth] = result;
  result]
```

Cone

Here we explore a right circular cone, oriented so that the point of the cone is upwards.

Accessing

```
assumptions[cone[h_, r_]] := h >= 0 && r >= 0
assumptions[cone[h_, α_, "apexangle"]] := FullSimplify[h >= 0 && α > 0 && α < π / 2]
assumptions[cone[h_, β_, "baseangle"]] := FullSimplify[assumptions[cone[h, complement[β], "apexangle"]]]
```

```
test @ assumptions[cone[h, α, "apexangle"]];
test @ assumptions[cone[h, β, "baseangle"]];
```

assumptions[cone[h, α, apexangle]] → $h \geq 0 \&\& \alpha > 0 \&\& 2\alpha < \pi$

assumptions[cone[h, β, baseangle]] → $h \geq 0 \&\& 2\beta < \pi \&\& \beta > 0$

```
radius[c : cone[h_, r_]] := r
radius[c : cone[h_, α_, "apexangle"]] := h Tan[α]
radius[c : cone[h_, β_, "baseangle"]] := h Cot[β]

height[c : cone[h_, r_]] := h
height[c : cone[h_, α_, "apexangle"]] := h
height[c : cone[h_, β_, "baseangle"]] := h
```

```
apexangle[c : cone[h_, r_]] := Assuming[assumptions[c], ArcTan[h, r]]
apexangle[c : cone[h_, α_, "apexangle"]] := α
apexangle[c : cone[h_, β_, "baseangle"]] := complement[baseangle[c]]
baseangle[c : cone[h_, r_]] := Assuming[assumptions[c], ArcTan[r, h]]
baseangle[c : cone[h_, α_, "apexangle"]] := complement[α]
baseangle[c : cone[h_, β_, "baseangle"]] := β
```

```
test @ apexangle[cone[h, r]];
test @ apexangle[cone[h, α, "apexangle"]];
test @ apexangle[cone[h, β, "baseangle"]];
test @ baseangle[cone[h, r]];
test @ baseangle[cone[h, α, "apexangle"]];
test @ baseangle[cone[h, β, "baseangle"]];
```

apexangle[cone[h, r]] → ArcTan[h, r]

apexangle[cone[h, α, apexangle]] → α

apexangle[cone[h, β, baseangle]] → $\frac{\pi}{2} - \beta$

baseangle[cone[h, r]] → ArcTan[r, h]

baseangle[cone[h, α, apexangle]] → $\frac{\pi}{2} - \alpha$

baseangle[cone[h, β, baseangle]] → β

Conversion

```
toCone[c : cone[h_, r_]] := c
toCone[c : cone[h_, α_, "apexangle"]] := cone[h, radius[c]]
toCone[c : cone[h_, β_, "baseangle"]] := cone[h, radius[c]]

toCartesian[c : cone[h_, r_]] := toCone @ c
toCartesian[c : cone[h_, α_, "apexangle"]] := toCone @ c
toCartesian[c : cone[h_, β_, "baseangle"]] := toCone @ c

toApexAngled[c : cone[h_, r_]] := cone[h, apexangle[c], "apexangle"]
toApexAngled[c : cone[h_, α_, "apexangle"]] := c
toApexAngled[c : cone[h_, β_, "baseangle"]] := cone[h, apexangle[c], "apexangle"]

toBaseAngled[c : cone[h_, r_]] := cone[h, baseangle[c], "baseangle"]
toBaseAngled[c : cone[h_, α_, "apexangle"]] := cone[h, baseangle[c], "baseangle"]
toBaseAngled[c : cone[h_, β_, "baseangle"]] := c

scaled[c : cone[h_, r_], factor_] := cone[h * factor, r * factor]
scaled[c : cone[h_, α_, "apexangle"], factor_] := toApexAngled @ scaled[toCartesian @ c, factor]
scaled[c : cone[h_, β_, "baseangle"], factor_] := toBaseAngled @ scaled[toCartesian @ c, factor]
```

```
test @ toCone[cone[h, r]];
test @ toCone[cone[h, α, "apexangle"]];
test @ toCone[cone[h, β, "baseangle"]];
test @ toApexAngled[cone[h, r]];
test @ toApexAngled[cone[h, α, "apexangle"]];
test @ toApexAngled[cone[h, β, "baseangle"]];
test @ toBaseAngled[cone[h, r]];
test @ toBaseAngled[cone[h, α, "apexangle"]];
test @ toBaseAngled[cone[h, β, "baseangle"]];
test @ scaled[cone[h, r], 2];
test @ scaled[cone[h, α, "apexangle"], 2];
test @ scaled[cone[h, β, "baseangle"], 2];
```

toCone[cone[h, r]] → cone[h, r]

toCone[cone[h, α, apexangle]] → cone[h, h Tan[α]]

toCone[cone[h, β, baseangle]] → cone[h, h Cot[β]]

toApexAngled[cone[h, r]] → cone[h, ArcTan[h, r], apexangle]

toApexAngled[cone[h, α, apexangle]] → cone[h, α, apexangle]

toApexAngled[cone[h, β, baseangle]] → cone$\left[h, \frac{\pi}{2} - \beta, \text{apexangle}\right]$

toBaseAngled[cone[h, r]] → cone[h, ArcTan[r, h], baseangle]

toBaseAngled[cone[h, α, apexangle]] → cone$\left[h, \frac{\pi}{2} - \alpha, \text{baseangle}\right]$

toBaseAngled[cone[h, β, baseangle]] → cone[h, β, baseangle]

scaled[cone[h, r], 2] → cone[2 h, 2 r]

scaled[cone[h, α, apexangle], 2] → cone[2 h, ArcTan[2 h, 2 h Tan[α]], apexangle]

scaled[cone[h, β, baseangle], 2] → cone[2 h, ArcTan[2 h Cot[β], 2 h], baseangle]

Volume

```
volume[c:cone[h_, r_]] := Pi r r h / 3
volume[c:cone[h_, α_, "apexangle"]] := volume @ toCartesian @ c
volume[c:cone[h_, β_, "baseangle"]] := volume @ toCartesian @ c
test @ volume[cone[h, r]];
test @ volume[cone[h, α, "apexangle"]];
test @ volume[cone[h, β, "baseangle"]];
```

volume[cone[h, r]] → $\frac{1}{3} h \pi r^2$

volume[cone[h, α, apexangle]] → $\frac{1}{3} h^3 \pi \operatorname{Tan}[\alpha]^2$

volume[cone[h, β, baseangle]] → $\frac{1}{3} h^3 \pi \operatorname{Cot}[\beta]^2$

Height and Depth

```
genericConeDepthFromVolume[] := Module[{c, cc, h, r, hh, vol, a, eqn, solns, soln},
  (* conjures up a soln with varaibles known to be free *)
  c = cone[h, r];
  cc = scaled[c, hh / h];
  a = assumptions[c] && assumptions[cc] && vol ≥ 0 ;
  eqn = FullSimplify[vol == volume[c] - volume[cc], a];
  solns = Assuming[a, Solve[eqn, hh]];
  soln = FullSimplify[h - (hh /. First @ solns), a];
  genericConeDepthFromVolume[] = {h, r, vol, soln}
 ]
test @ genericConeDepthFromVolume[];
```

genericConeDepthFromVolume[] → $\left\{h\$2523, r\$2523, vol\$2523, h\$2523 - \left(\frac{h\$2523}{r\$2523}\right)^{2/3} \left(h\$2523\, r\$2523^2 - \frac{3\, vol\$2523}{\pi}\right)^{1/3}\right\}$

```
depthFromVolume[c:cone[h_, r_], v_] := Module[{hh, rr, vol, soln},
  {hh, rr, vol, soln} = genericConeDepthFromVolume[];
  (soln /. {hh → h, rr → r, vol → v}) // FullSimplify
 ]
depthFromVolume[c:cone[h_, α_, "apexangle"], v_] := depthFromVolume[toCartesian @ c, v]
depthFromVolume[c:cone[h_, β_, "baseangle"], v_] := depthFromVolume[toCartesian @ c, v]

test @ depthFromVolume[cone[h, r], volume];
test @ depthFromVolume[cone[h, α, "apexangle"], volume];
test @ depthFromVolume[cone[h, β, "baseangle"], volume];
```

depthFromVolume[cone[h, r], volume] → $h - \left(\frac{h}{r}\right)^{2/3} \left(h\, r^2 - \frac{3\, volume}{\pi}\right)^{1/3}$

depthFromVolume[cone[h, α, apexangle], volume] → $h - \operatorname{Cot}[\alpha]^{2/3} \left(-\frac{3\, volume}{\pi} + h^3 \operatorname{Tan}[\alpha]^2\right)^{1/3}$

depthFromVolume[cone[h, β, baseangle], volume] → $h - \left(-\frac{3\, volume}{\pi} + h^3 \operatorname{Cot}[\beta]^2\right)^{1/3} \operatorname{Tan}[\beta]^{2/3}$

```
volumeFromDepth[c : cone[h_, r_], depth_] := genericVolumeFromDepthUsingInverse[cone[hh, rr], dd] /. {hh → h, rr → r, dd → depth}
volumeFromDepth[c : cone[h_, α_, "apexangle"], v_] := volumeFromDepth[toCartesian @ c, v]
volumeFromDepth[c : cone[h_, β_, "baseangle"], v_] := volumeFromDepth[toCartesian @ c, v]

test @ volumeFromDepth[cone[h, r], depth];
test @ volumeFromDepth[cone[h, α, "apexangle"], depth];
test @ volumeFromDepth[cone[h, β, "baseangle"], depth];
```

volumeFromDepth[cone[h, r], depth] → $\frac{\text{depth}\left(\text{depth}^2 - 3\,\text{depth}\,h + 3\,h^2\right)\pi\,r^2}{3\,h^2}$

volumeFromDepth[cone[h, α, apexangle], depth] → $\frac{1}{3}\text{depth}\left(\text{depth}^2 - 3\,\text{depth}\,h + 3\,h^2\right)\pi\,\text{Tan}[\alpha]^2$

volumeFromDepth[cone[h, β, baseangle], depth] → $\frac{1}{3}\text{depth}\left(\text{depth}^2 - 3\,\text{depth}\,h + 3\,h^2\right)\pi\,\text{Cot}[\beta]^2$

```
radiusFromDepth[c : cone[h_, α_, "apexangle"], depth_] := Block[{hRemaining, eqn, result},
  (*hRemaining = h -depth;
  eqn = result / hRemaining == Tan[α];
  result /. First @ Solve[eqn, result]*)
  (h - depth) Tan[α]]
radiusFromDepth[c : cone[h_, r_], depth_] := radiusFromDepth[toApexAngled[c], depth]
radiusFromDepth[c : cone[h_, β_, "baseangle"], depth_] := radiusFromDepth[toApexAngled[c], depth]

test @ radiusFromDepth[cone[h, α, "apexangle"], depth];
test @ radiusFromDepth[cone[h, β, "baseangle"], depth];
test @ radiusFromDepth[cone[h, r], depth];
```

radiusFromDepth[cone[h, α, apexangle], depth] → (-depth + h) Tan[α]

radiusFromDepth[cone[h, β, baseangle], depth] → (-depth + h) Cot[β]

radiusFromDepth[cone[h, r], depth] → $\frac{(-\text{depth} + h)\,r}{h}$

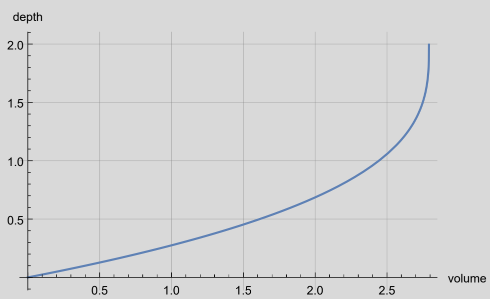
Testing

```
example = cone[2, π / 6, "apexangle"]
{ volume[example], volume[example] // N }
expr = test @ depthFromVolume[example, v];
Plot[expr, {v, 0, volume[example]}, AxesLabel → {"volume", "depth"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ volumeFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "volume"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ radiusFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "radius"}, AxesOrigin → {0, 0}, GridLines → Automatic]
```

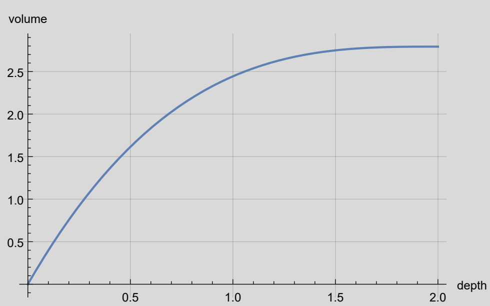
$\text{cone}\left[2, \frac{\pi}{6}, \text{apexangle}\right]$

$\left\{\frac{8\pi}{9}, 2.79253\right\}$

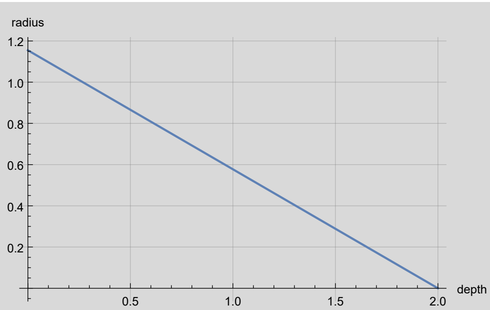
$\text{depthFromVolume}[\text{example}, v] \to 2 - \left(8 - \frac{9\,v}{\pi}\right)^{1/3}$



$\text{volumeFromDepth}[\text{example}, \text{depth}] \to \frac{1}{9}\,\text{depth}\,\left(12 - 6\,\text{depth} + \text{depth}^2\right)\,\pi$



$\text{radiusFromDepth}[\text{example}, \text{depth}] \to \frac{2 - \text{depth}}{\sqrt{3}}$



Inverted Cone

Here we explore an inversion of a right circular cone: the point is downwards, as it would be for ice-cream.

Construction & Conversion

```
toCone[c : invertedCone[h_, r_]] := invert @ c
toCone[c : invertedCone[h_, α_, "apexangle"]] := invert @ c
toCone[c : invertedCone[h_, β_, "baseangle"]] := invert @ c

toCartesian[c : invertedCone[h_, r_]] := invert @ toCartesian @ invert @ c
toCartesian[c : invertedCone[h_, α_, "apexangle"]] := invert @ toCartesian @ invert @ c
toCartesian[c : invertedCone[h_, β_, "baseangle"]] := invert @ toCartesian @ invert @ c

invert[c : invertedCone[h_, r_]] := cone[h, r]
invert[c : invertedCone[h_, α_, "apexangle"]] := cone[h, α, "apexangle"]
invert[c : invertedCone[h_, β_, "baseangle"]] := cone[h, β, "baseangle"]

invert[c : cone[h_, r_]] := invertedCone[h, r]
invert[c : cone[h_, α_, "apexangle"]] := invertedCone[h, α, "apexangle"]
invert[c : cone[h_, β_, "baseangle"]] := invertedCone[h, β, "baseangle"]

scaled[c : invertedCone[h_, r_], factor_] := invertedCone[h * factor, r * factor]
scaled[c : invertedCone[h_, α_, "apexangle"], factor_] := toApexAngled @ scaled[toCartesian @ c, factor]
scaled[c : invertedCone[h_, β_, "baseangle"], factor_] := toBaseAngled @ scaled[toCartesian @ c, factor]
```

```
test @ scaled[invertedCone[h, r], 2];
test @ scaled[invertedCone[h, α, "apexangle"], 2] ;
test @ scaled[invertedCone[h, β, baseangle], 2];
```

```
scaled[invertedCone[h, r], 2] → invertedCone[2 h, 2 r]
```

```
scaled[invertedCone[h, α, apexangle], 2] → toApexAngled[invertedCone[2 h, 2 h Tan[α]]]
```

```
scaled[invertedCone[h, β, baseangle], 2] → scaled[invertedCone[h, β, baseangle], 2]
```

Accessing

```
assumptions[c : invertedCone[h_, r_]] := assumptions[toCone @ c]
assumptions[c : invertedCone[h_, α_, "apexangle"]] := assumptions[toCone @ c]
assumptions[c : invertedCone[h_, β_, "baseangle"]] := assumptions[toCone @ c]
test @ assumptions[invertedCone[h, α, "apexangle"]];
test @ assumptions[invertedCone[h, β, "baseangle"]];
```

```
assumptions[invertedCone[h, α, apexangle]] → h ≥ 0 && α > 0 && 2 α < π
```

```
assumptions[invertedCone[h, β, baseangle]] → h ≥ 0 && 2 β < π && β > 0
```

```
radius[c : invertedCone[h_, r_]] := r
radius[c : invertedCone[h_, α_, "apexangle"]] := radius @ invert @ c
radius[c : invertedCone[h_, β_, "baseangle"]] := radius @ invert @ c

height[c : invertedCone[h_, r_]] := h
height[c : invertedCone[h_, α_, "apexangle"]] := h
height[c : invertedCone[h_, β_, "baseangle"]] := h
```

```
apexangle[c : invertedCone[h_, r_]] := Assuming[assumptions[c], ArcTan[h, r]]
apexangle[c : invertedCone[h_, α_, "apexangle"]] := α
apexangle[c : invertedCone[h_, β_, "baseangle"]] := complement[baseangle[c]]
baseangle[c : invertedCone[h_, r_]] := Assuming[assumptions[c], ArcTan[r, h]]
baseangle[c : invertedCone[h_, α_, "apexangle"]] := complement[α]
baseangle[c : invertedCone[h_, β_, "baseangle"]] := β
```

```
test @ apexangle[invertedCone[h, r]];
test @ apexangle[invertedCone[h, α, "apexangle"]];
test @ apexangle[invertedCone[h, β, "baseangle"]];
test @ baseangle[invertedCone[h, r]];
test @ baseangle[invertedCone[h, α, "apexangle"]];
test @ baseangle[invertedCone[h, β, "baseangle"]];
```

apexangle[invertedCone[h, r]] → ArcTan[h, r]

apexangle[invertedCone[h, α, apexangle]] → α

apexangle[invertedCone[h, β, baseangle]] → $\frac{\pi}{2} - \beta$

baseangle[invertedCone[h, r]] → ArcTan[r, h]

baseangle[invertedCone[h, α, apexangle]] → $\frac{\pi}{2} - \alpha$

baseangle[invertedCone[h, β, baseangle]] → β

Conversion Redux

```
toInvertedCone[c : invertedCone[h_, r_]] := c
toInvertedCone[c : invertedCone[h_, α_, "apexangle"]] := invertedCone[h, h Tan[α]]
toInvertedCone[c : invertedCone[h_, β_, "baseangle"]] := toInvertedCone[toApexAngled[c]]

toCartesian[c : invertedCone[h_, r_]] := toInvertedCone @ c
toCartesian[c : invertedCone[h_, α_, "apexangle"]] := toInvertedCone @ c
toCartesian[c : invertedCone[h_, β_, "baseangle"]] := toInvertedCone @ c

toApexAngled[c : invertedCone[h_, r_]] := invertedCone[h, apexangle[c], "apexangle"]
toApexAngled[c : invertedCone[h_, α_, "apexangle"]] := c
toApexAngled[c : invertedCone[h_, β_, "baseangle"]] := invertedCone[h, apexangle[c], "apexangle"]

toBaseAngled[c : invertedCone[h_, r_]] := invertedCone[h, baseangle[c], "baseangle"]
toBaseAngled[c : invertedCone[h_, α_, "apexangle"]] := invertedCone[h, baseangle[c], "baseangle"]
toBaseAngled[c : invertedCone[h_, β_, "baseangle"]] := c
```

```
test @ toInvertedCone[invertedCone[h, r]];
test @ toInvertedCone[invertedCone[h, α, "apexangle"]];
test @ toInvertedCone[invertedCone[h, β, "baseangle"]];
test @ toApexAngled[invertedCone[h, r]];
test @ toApexAngled[invertedCone[h, α, "apexangle"]];
test @ toApexAngled[invertedCone[h, β, "baseangle"]];
test @ toBaseAngled[invertedCone[h, r]];
test @ toBaseAngled[invertedCone[h, α, "apexangle"]];
test @ toBaseAngled[invertedCone[h, β, "baseangle"]];
```

toInvertedCone[invertedCone[h, r]] → invertedCone[h, r]

toInvertedCone[invertedCone[h, α, apexangle]] → invertedCone[h, h Tan[α]]

toInvertedCone[invertedCone[h, β, baseangle]] → invertedCone[h, h Cot[β]]

toApexAngled[invertedCone[h, r]] → invertedCone[h, ArcTan[h, r], apexangle]

toApexAngled[invertedCone[h, α, apexangle]] → invertedCone[h, α, apexangle]

toApexAngled[invertedCone[h, β, baseangle]] → invertedCone[h, $\frac{\pi}{2} - \beta$, apexangle]

toBaseAngled[invertedCone[h, r]] → invertedCone[h, ArcTan[r, h], baseangle]

toBaseAngled[invertedCone[h, α, apexangle]] → invertedCone[h, $\frac{\pi}{2} - \alpha$, baseangle]

toBaseAngled[invertedCone[h, β, baseangle]] → invertedCone[h, β, baseangle]

Volume

```
volume[c : invertedCone[h_, r_]] := volume @ toCone @ c
volume[c : invertedCone[h_, α_, "apexangle"]] := volume @ toCone @ c
volume[c : invertedCone[h_, β_, "baseangle"]] := volume @ toCone @ c
test @ volume[invertedCone[h, r]];
test @ volume[invertedCone[h, α, "apexangle"]];
test @ volume[invertedCone[h, β, "baseangle"]];
```

volume[invertedCone[h, r]] → $\frac{1}{3} h \pi r^2$

volume[invertedCone[h, α, apexangle]] → $\frac{1}{3} h^3 \pi \operatorname{Tan}[\alpha]^2$

volume[invertedCone[h, β, baseangle]] → $\frac{1}{3} h^3 \pi \operatorname{Cot}[\beta]^2$

Height and Depth

```
genericInvertedConeDepthFromVolume[] := Module[{c, h, α, hh, vol, a, eqn, solns, soln},
  c = invertedCone[h, α, "apexangle"];
  a = assumptions[c] && vol ≥ 0;
  eqn = FullSimplify[vol == volume[c], a];
  solns = Assuming[a, Solve[eqn, h]];
  soln = FullSimplify[h /. solns[[2]], a];
  genericInvertedConeDepthFromVolume[] = {α, vol, soln}
 ]
test @ genericInvertedConeDepthFromVolume[];
```

genericInvertedConeDepthFromVolume[] → $\left\{\alpha\$4594, \text{vol}\$4594, \left(\frac{3}{\pi}\right)^{1/3} \left(\text{vol}\$4594 \operatorname{Cot}[\alpha\$4594]^2\right)^{1/3}\right\}$

```
depthFromVolume[c : invertedCone[ignored_, α_, "apexangle"], v_] := Module[{αα, vol, soln},
  {αα, vol, soln} = genericInvertedConeDepthFromVolume[];
  (soln /. {αα → α, vol → v}) // FullSimplify
 ]
depthFromVolume[c : invertedCone[h_, r_], v_] := depthFromVolume[toApexAngled @ c, v]
depthFromVolume[c : invertedCone[h_, β_, "baseangle"], v_] := depthFromVolume[toApexAngled @ c, v]

test @ depthFromVolume[invertedCone[ignored, α, "apexangle"], volume];
test @ depthFromVolume[invertedCone[h, r], volume];
test @ depthFromVolume[invertedCone[h, β, "baseangle"], volume];
```

depthFromVolume[invertedCone[ignored, α, apexangle], volume] → $\left(\frac{3}{\pi}\right)^{1/3} \left(\text{volume} \operatorname{Cot}[\alpha]^2\right)^{1/3}$

depthFromVolume[invertedCone[h, r], volume] → $\left(\frac{3}{\pi}\right)^{1/3} \left(\frac{h^2\, \text{volume}}{r^2}\right)^{1/3}$

depthFromVolume[invertedCone[h, β, baseangle], volume] → $\left(\frac{3}{\pi}\right)^{1/3} \left(\text{volume} \operatorname{Tan}[\beta]^2\right)^{1/3}$

```
volumeFromDepth[c : invertedCone[h_, α_, "apexangle"], depth_] :=
 genericVolumeFromDepthUsingInverse[invertedCone[hh, αα, "apexangle"], dd] /. {hh → h, αα → α, dd → depth}
volumeFromDepth[c : invertedCone[h_, β_, "baseangle"], depth_] :=
 genericVolumeFromDepthUsingInverse[invertedCone[hh, ββ, "baseangle"], dd] /. {hh → h, ββ → β, dd → depth}
volumeFromDepth[c : invertedCone[h_, r_], depth_] :=
 genericVolumeFromDepthUsingInverse[invertedCone[hh, rr], dd] /. {hh → h, rr → r, dd → depth}
test @ volumeFromDepth[invertedCone[h, α, "apexangle"], depth];
test @ volumeFromDepth[invertedCone[h, β, "baseangle"], depth];
test @ volumeFromDepth[invertedCone[h, r], depth];
```

volumeFromDepth[invertedCone[h, α, apexangle], depth] → $\frac{1}{3}\text{depth}^3\,\pi\,\text{Tan}[\alpha]^2$

volumeFromDepth[invertedCone[h, β, baseangle], depth] → $\frac{1}{3}\text{depth}^3\,\pi\,\text{Cot}[\beta]^2$

volumeFromDepth[invertedCone[h, r], depth] → $\frac{\text{depth}^3\,\pi\,\text{r}^2}{3\,\text{h}^2}$

```
radiusFromDepth[c : invertedCone[h_, α_, "apexangle"], depth_] := Block[{eqn, result},
  (*eqn = result / depth == Tan[α];
  result /. First @ Solve[eqn, result]*)
  depth Tan[α]]
radiusFromDepth[c : invertedCone[h_, r_], depth_] := radiusFromDepth[toApexAngled[c], depth]
radiusFromDepth[c : invertedCone[h_, β_, "baseangle"], depth_] := radiusFromDepth[toApexAngled[c], depth]

test @ radiusFromDepth[invertedCone[h, α, "apexangle"], depth];
test @ radiusFromDepth[invertedCone[h, β, "baseangle"], depth];
test @ radiusFromDepth[invertedCone[h, r], depth];
```

radiusFromDepth[invertedCone[h, α, apexangle], depth] → depth Tan[α]

radiusFromDepth[invertedCone[h, β, baseangle], depth] → depth Cot[β]

radiusFromDepth[invertedCone[h, r], depth] → $\frac{\text{depth r}}{\text{h}}$

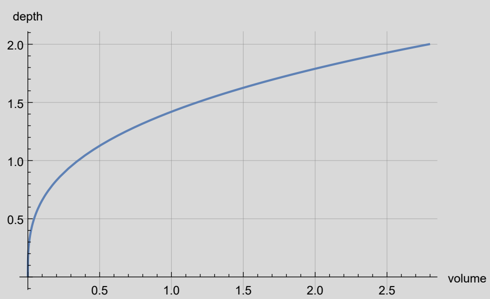
Testing

```
example = invertedCone[2, π / 6, "apexangle"]
{ volume[example], volume[example] // N }
expr = test @ depthFromVolume[example, v];
Plot[expr, {v, 0, volume[example]}, AxesLabel → {"volume", "depth"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ volumeFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "volume"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ radiusFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "radius"}, AxesOrigin → {0, 0}, GridLines → Automatic]
```

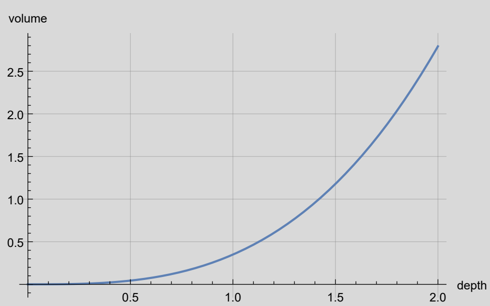
invertedCone$\left[2, \frac{\pi}{6}, \text{apexangle}\right]$

$\left\{\frac{8\,\pi}{9}, 2.79253\right\}$

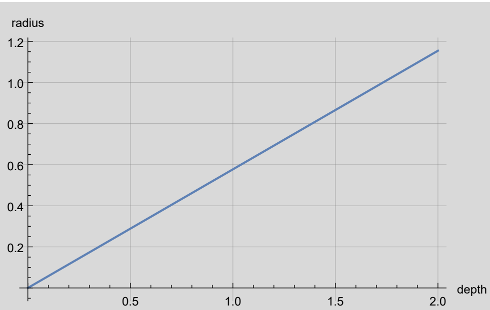
depthFromVolume[example, v] $\to \frac{3^{2/3}\,v^{1/3}}{\pi^{1/3}}$



volumeFromDepth[example, depth] $\to \frac{\text{depth}^3\,\pi}{9}$



radiusFromDepth[example, depth] $\to \frac{\text{depth}}{\sqrt{3}}$

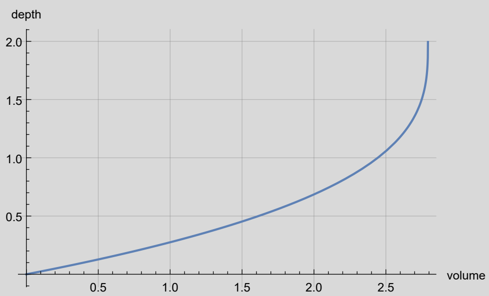


```
example = cone[2, π / 6, "apexangle"]
{ volume[example], volume[example] // N }
expr = test @ depthFromVolume[example, v];
Plot[expr, {v, 0, volume[example]}, AxesLabel → {"volume", "depth"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ volumeFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "volume"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ radiusFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "radius"}, AxesOrigin → {0, 0}, GridLines → Automatic]
```

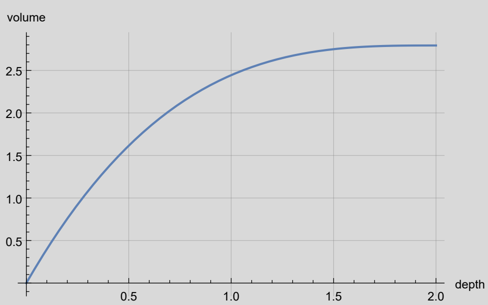
$\text{cone}\left[2, \frac{\pi}{6}, \text{apexangle}\right]$

$\left\{\frac{8\pi}{9}, 2.79253\right\}$

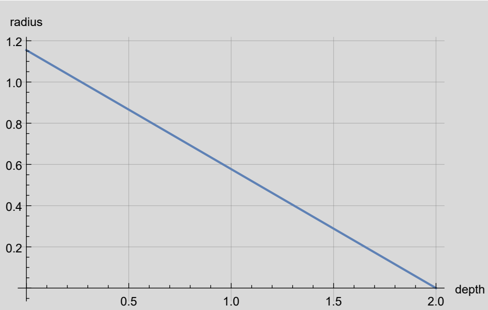
$\text{depthFromVolume}[\text{example}, v] \to 2 - \left(8 - \frac{9 v}{\pi}\right)^{1/3}$



$\text{volumeFromDepth}[\text{example}, \text{depth}] \to \frac{1}{9} \text{depth} \left(12 - 6\,\text{depth} + \text{depth}^2\right) \pi$



$\text{radiusFromDepth}[\text{example}, \text{depth}] \to \frac{2 - \text{depth}}{\sqrt{3}}$



Cylinder

Cylinders are easy, simple shapes.

Accessing

```
assumptions[cylinder[h_, r_]] := h >= 0 && r >= 0
```

```
test @ assumptions[cylinder[h, r]];
```

```
assumptions[cylinder[h, r]] → h ≥ 0&&r ≥ 0
```

```
emptyCylinder[] := cylinder[0, 0]
height[c : cylinder[h_, r_]] := h
radius[c : cylinder[h_, r_]] := r
```

```
toCartesian[c : cylinder[h_, r_]] := c
toApexAngled[c : cylinder[h_, r_]] := c
toBaseAngled[c : cylinder[h_, r_]] := c
```

Volume

```
volume[cylinder[h_, r_]] := Pi r r h
test @ volume[cylinder[h, r]];
test @ volume @ emptyCylinder[];
```

volume[cylinder[h, r]] → $h \pi r^2$

volume[emptyCylinder[]] → 0

Height and Depth

```
depthFromVolume[c : cylinder[_, 0], v_] := 0
depthFromVolume[c : cylinder[0, _], v_] := 0
depthFromVolume[c : cylinder[_, r_], v_] := Module[{hh}, hh /. First @ Solve[v == volume[cylinder[hh, r]], hh]]
test @ depthFromVolume[cylinder[ignored, r], volume];
test @ depthFromVolume[cylinder[1, 2], volume];
test @ depthFromVolume[emptyCylinder[], volume];
```

depthFromVolume[cylinder[ignored, r], volume] → $\frac{\text{volume}}{\pi r^2}$

depthFromVolume[cylinder[1, 2], volume] → $\frac{\text{volume}}{4 \pi}$

depthFromVolume[emptyCylinder[], volume] → 0

```
volumeFromDepth[c : cylinder[h_, r_], depth_] := genericVolumeFromDepthUsingInverse[cylinder[hh, rr], dd] /. {hh → h, rr → r, dd → depth}
test @ volumeFromDepth[cylinder[h, r], depth];
```

volumeFromDepth[cylinder[h, r], depth] → $\text{depth}\, \pi r^2$

```
radiusFromDepth[c : cylinder[h_, r_], depth_] := r
```

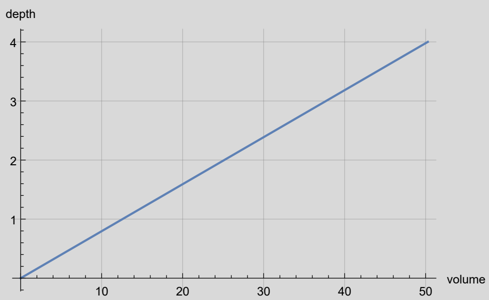
Testing

```
example = cylinder[4, 2]
{ volume[example], volume[example] // N }
expr = test @ depthFromVolume[example, v];
Plot[expr, {v, 0, volume[example]}, AxesLabel → {"volume", "depth"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ volumeFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "volume"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ radiusFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "radius"}, AxesOrigin → {0, 0}, GridLines → Automatic]
```

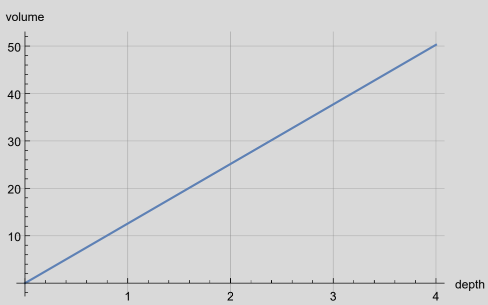
```
cylinder[4, 2]
```

$\{16\,\pi,\ 50.2655\}$

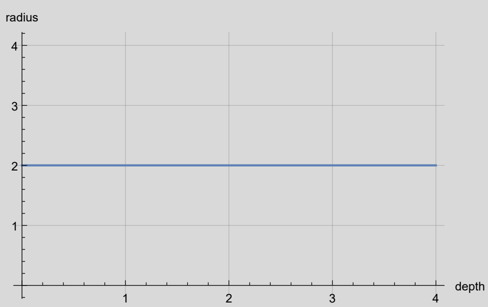
depthFromVolume[example, v] → $\frac{v}{4\,\pi}$



volumeFromDepth[example, depth] → 4 depth π



radiusFromDepth[example, depth] → 2



Right Conical Frustum

A right conical frustum is a truncated right circular cone, where the plane of truncation is parallel to the plane of the cone base (and at right angles to the cone axis).

Accessing

```
assumptions[frustum[h_, rbig_, rsmall_]] := h ≥ 0 && rbig ≥ 0 && rsmall ≥ 0 && rbig > rsmall
assumptions[frustum[h_, rbig_, α_, "apexangle"]] := FullSimplify @ assumptions[frustum[h, rbig, complement[α], "baseangle"]]
assumptions[frustum[h_, rbig_, β_, "baseangle"]] := FullSimplify[h ≥ 0 && rbig ≥ 0 && β > 0 && β < π / 2]
```

```
test @ assumptions[frustum[h, rbig, α, "apexangle"]];
test @ assumptions[frustum[h, rbig, β, "baseangle"]];
```

```
assumptions[frustum[h, rbig, α, apexangle]] → h ≥ 0 && rbig ≥ 0 && 2 α < π && α > 0
```

```
assumptions[frustum[h, rbig, β, baseangle]] → h ≥ 0 && rbig ≥ 0 && β > 0 && 2 β < π
```

```
apexangle[f : frustum[h_, rbig_, α_, "apexangle"]] := α
apexangle[f : frustum[h_, rbig_, β_, "baseangle"]] := complement[baseangle[f]]
apexangle[f : frustum[h_, rbig_, rsmall_]] := Assuming[assumptions[f], ArcTan[h, rbig - rsmall]]

baseangle[f : frustum[h_, rbig_, α_, "apexangle"]] := complement[ apexangle[f]]
baseangle[f : frustum[h_, rbig_, β_, "baseangle"]] := β
baseangle[f : frustum[h_, rbig_, rsmall_]] := Assuming[assumptions[f], ArcTan[rbig - rsmall, h]]

baseangle[f : frustum[h_, rbig_, rbig_ - h_ Cot[β_]]] := β

test @ apexangle[frustum[h, rbig, rsmall]];
test @ baseangle[frustum[h, rbig, rsmall]];
test @ { baseangle[frustum[1, 3, 2]], baseangle[frustum[Sqrt[3], 2, 1]]};
```

```
apexangle[frustum[h, rbig, rsmall]] → ArcTan[h, rbig - rsmall]
```

```
baseangle[frustum[h, rbig, rsmall]] → ArcTan[rbig - rsmall, h]
```

{baseangle[frustum[1, 3, 2]], baseangle[frustum[$\sqrt{3}$, 2, 1]]} → $\left\{\frac{\pi}{4}, \frac{\pi}{3}\right\}$

```
Solve[(rbig - rsmall) / h == Tan[α], rsmall]
Solve[(rbig - rsmall) / h == Tan[α], rbig]
```

```
{{rsmall → rbig - h Tan[α]}}
```

```
{{rbig → rsmall + h Tan[α]}}
```

```
rbig[h_, rsmall_, α_, "apexangle"] := rsmall + h Tan[α]
rsmall[h_, rbig_, α_, "apexangle"] := rbig - h Tan[α]
rbig[h_, rsmall_, β_, "baseangle"] := rbig[h, rsmall, complement[β], "apexangle"]
rsmall[h_, rbig_, β_, "baseangle"] := rsmall[h, rsmall, complement[β], "apexangle"]
```

```
height[f : frustum[h_, rbig_, α_, "apexangle"]] := h
height[f : frustum[h_, rbig_, β_, "baseangle"]] := h
height[f : frustum[h_, rbig_, rsmall_]] := h
```

```
rbig[f : frustum[h_, rbig_, α_, "apexangle"]] := rbig
rbig[f : frustum[h_, rbig_, β_, "baseangle"]] := rbig
rbig[f : frustum[h_, rbig_, rsmall_]] := rbig
```

```
Tan[α] / Cot[complement[α]] == 1
rsmall[f : frustum[h_, rbig_, α_, "apexangle"]] := Assuming[assumptions[f], rsmall[h, rbig, α, "apexangle"]]
rsmall[f : frustum[h_, rbig_, β_, "baseangle"]] := Assuming[assumptions[f], rsmall[h, rbig, β, "baseangle"]]
rsmall[f : frustum[h_, rbig_, rsmall_]] := rsmall
rsmall[f : frustum[h_, rbig_, ArcTan[rbig_ - rsmall_, h_], "baseangle"]] := rsmall
test @ rsmall[frustum[h, rbig, α, "apexangle"]];
test @ rsmall[frustum[h, rbig, β, "baseangle"]];
test @ rsmall[frustum[h, rbig, rsmall]];
```

```
True
```

```
rsmall[frustum[h, rbig, α, apexangle]] → rbig - h Tan[α]
```

```
rsmall[frustum[h, rbig, β, baseangle]] → rsmall - h Cot[β]
```

```
rsmall[frustum[h, rbig, rsmall]] → rsmall
```

Construction & Conversion

```
toFrustum[f : frustum[h_, rbig_, α_, "apexangle"]] := frustum[h, rbig, rsmall[f]]
toFrustum[f : frustum[h_, rbig_, β_, "baseangle"]] := frustum[h, rbig, rsmall[f]]
toFrustum[f : frustum[h_, rbig_, rsmall_]] := f

toCartesian[f : frustum[h_, rbig_, α_, "apexangle"]] := toFrustum @ f
toCartesian[f : frustum[h_, rbig_, β_, "baseangle"]] := toFrustum @ f
toCartesian[f : frustum[h_, rbig_, rsmall_]] := toFrustum @ f

toApexAngled[f : frustum[h_, rbig_, α_, "apexangle"]] := f
toApexAngled[f : frustum[h_, rbig_, β_, "baseangle"]] := frustum[h, rbig, complement[β], "apexangle"]
toApexAngled[f : frustum[h_, rbig_, rsmall_]] := frustum[h, rbig, apexangle[f], "apexangle"]

toBaseAngled[f : frustum[h_, rbig_, α_, "apexangle"]] := frustum[h, rbig, complement[α], "baseangle"]
toBaseAngled[f : frustum[h_, rbig_, β_, "baseangle"]] := f
toBaseAngled[f : frustum[h_, rbig_, rsmall_]] := frustum[h, rbig, baseangle[f], "baseangle"]
```

```
test @ toCartesian @ frustum[h, rbig, β, "baseangle"];
test @ toBaseAngled @ %;
test @ toApexAngled @ %%;
test @ toFrustum @ %;
test @ toBaseAngled @ %%;
```

```
toCartesian[frustum[h, rbig, β, baseangle]] → frustum[h, rbig, rsmall - h Cot[β]]
```

```
toBaseAngled[%] → frustum[h, rbig, ArcTan[rbig - rsmall + h Cot[β], h], baseangle]
```

```
toApexAngled[%%] → frustum[h, rbig, ArcTan[h, rbig - rsmall + h Cot[β]], apexangle]
```

```
toFrustum[%] → frustum[h, rbig, rsmall - h Cot[β]]
```

toBaseAngled[%%] → frustum$\left[h, rbig, \frac{\pi}{2} - \text{ArcTan}[h, rbig - rsmall + h\,\text{Cot}[\beta]], baseangle\right]$

```
test @ toBaseAngled @ frustum[h, rbig, rsmall];
test @ toCartesian @ %;
```

```
toBaseAngled[frustum[h, rbig, rsmall]] → frustum[h, rbig, ArcTan[rbig - rsmall, h], baseangle]
```

```
toCartesian[%] → frustum[h, rbig, rsmall]
```

Volume

```
genericConeHeightCartesianFrustum[] := Module[{f, h, rbig, rsmall, eqn, ch},
  f = frustum[h, rbig, rsmall];
  eqn = ch / rbig == h / (rbig - rsmall);
  genericConeHeightCartesianFrustum[] = {h, rbig, rsmall, ch /. First @ Solve[eqn, ch]}
 ]

coneHeight[f : frustum[h_, rbig_, α_, "apexangle"]] := rbig / Tan[α]
coneHeight[f : frustum[h_, rbig_, β_, "baseangle"]] := rbig / Cot[β]
coneHeight[f : frustum[h_, rbig_, rsmall_]] := Module[{hh, rrbig, rrsmall, ch},
  {hh, rrbig, rrsmall, ch} = genericConeHeightCartesianFrustum[];
  ch /. {hh → h, rrbig → rbig, rrsmall → rsmall}
 ]
test @ coneHeight[frustum[h, rbig, α, "apexangle"]];
test @ coneHeight[frustum[h, rbig, β, "baseangle"]];
test @ toApexAngled @ frustum[h, rbig, β, "baseangle"];
test @ coneHeight @ %;
test @ coneHeight[frustum[h, rbig, rsmall]];
test @ coneHeight[frustum[1, 3, 2]];
```

coneHeight[frustum[h, rbig, α, apexangle]] → rbig Cot[α]

coneHeight[frustum[h, rbig, β, baseangle]] → rbig Tan[β]

toApexAngled[frustum[h, rbig, β, baseangle]] → frustum$\left[h, rbig, \frac{\pi}{2} - \beta, apexangle\right]$

coneHeight[%] → rbig Tan[β]

coneHeight[frustum[h, rbig, rsmall]] → $\frac{h\ rbig}{rbig - rsmall}$

coneHeight[frustum[1, 3, 2]] → 3

```
fullCone[f : frustum[h_, rbig_, α_, "apexangle"]] := cone[coneHeight[f], α, "apexangle"]
fullCone[f : frustum[h_, rbig_, β_, "baseangle"]] := fullCone @ toApexAngled @ f
fullCone[f : frustum[h_, rbig_, rsmall_]] := cone[coneHeight[f], rbig]
```

```
topCone[f : frustum[h_, rbig_, α_, "apexangle"]] := cone[coneHeight[f] - h, α, "apexangle"]
topCone[f : frustum[h_, rbig_, β_, "baseangle"]] := topCone @ toApexAngled @ f
topCone[f : frustum[h_, rbig_, rsmall_]] := Module[{full, eqn, scale, result},
  full = fullCone[f];
  result = scaled[full, scale];
  eqn = radius[result] == rsmall;
  result /. First @ Solve[eqn, scale]
 ]
test @ topCone[frustum[h, rbig, rsmall]];
```

topCone[frustum[h, rbig, rsmall]] → cone$\left[\frac{h\ rsmall}{rbig - rsmall}, rsmall\right]$

```
volume[f : frustum[h_, rbig_, rsmall_]] := volume[fullCone[f]] - volume[topCone[f]] // FullSimplify
volume[f : frustum[h_, rbig_, α_, "apexangle"]] := volume[fullCone[f]] - volume[topCone[f]] // FullSimplify
volume[f : frustum[h_, rbig_, β_, "baseangle"]] := volume @ toApexAngled[f]
```

```
(* compare to textbook answer 1/3 h π (r1^2+r1 r2+r2^2) *)
test @ volume[frustum[h, r1, r2]];
test @ volume[frustum[h, r, α, "apexangle"]];
test @ volume[toFrustum @ frustum[h, r, α, "apexangle"]];
% / %% // FullSimplify
test @ volume[frustum[h, r, β, "baseangle"]];
```

volume[frustum[h, r1, r2]] → $\frac{1}{3} h \pi \left(r1^2 + r1\, r2 + r2^2\right)$

volume[frustum[h, r, α, apexangle]] → $\frac{1}{3} h \pi \left(3 r^2 + h \operatorname{Tan}[\alpha] (-3 r + h \operatorname{Tan}[\alpha])\right)$

volume[toFrustum[frustum[h, r, α, apexangle]]] → $\frac{1}{3} \pi \operatorname{Cot}[\alpha] \left(r^3 - (r - h \operatorname{Tan}[\alpha])^3\right)$

1

volume[frustum[h, r, β, baseangle]] → $\frac{1}{3} h \pi \left(3 r^2 + h \operatorname{Cot}[\beta] (-3 r + h \operatorname{Cot}[\beta])\right)$

Height and Depth: Angled

```
genericFrustumDepthFromVolumeApex[] := Module[{f, h, rbig, α, vol, a, eqn, solns, depth},
  (* conjures up a soln with varaibles known to be free *)
  f = frustum[h, rbig, α, "apexangle"];
  a = assumptions[f] && vol ≥ 0;
  eqn = FullSimplify[vol == volume[f], a];
  solns = Assuming[a, Solve[eqn, h]];
  depth = FullSimplify[h /. First @ solns, a];
  genericFrustumDepthFromVolume1[] = {h, rbig, α, vol, depth}
 ]
test @ genericFrustumDepthFromVolumeApex[];
```

genericFrustumDepthFromVolumeApex[] → {h\$6868, rbig\$6868, α\$6868, vol\$6868, $\operatorname{Cot}[\alpha\$6868] \left(rbig\$6868 - \left(rbig\$6868^3 - \frac{3\, vol\$6868 \operatorname{Tan}[\alpha\$6868]}{\pi}\right)^{1/3}\right)$}

```
depthFromVolume[f : frustum[ignored_, rbig_, α_, "apexangle"], vol_] := Module[{hh, rr, αα, vv, eqn, depth},
  {hh, rr, αα, vv, depth} = genericFrustumDepthFromVolumeApex[];
  depth /. {rr → rbig, αα → α, vv → vol}
 ]
generalApexFrustum = frustum[h, rbig, α, "apexangle"]
test @ depthFromVolume[generalApexFrustum, vol];
```

frustum[h, rbig, α, apexangle]

depthFromVolume[generalApexFrustum, vol] → $\operatorname{Cot}[\alpha] \left(rbig - \left(rbig^3 - \frac{3\, vol \operatorname{Tan}[\alpha]}{\pi}\right)^{1/3}\right)$

```
depthFromVolume[f : frustum[ignored_, rbig_, β_, "baseangle"], vol_] := Module[{hh, rr, αα, vv, eqn, soln},
  {hh, rr, αα, vv, soln} = genericFrustumDepthFromVolumeApex[];
  soln /. {rr → rbig, αα → apexangle[f], vv → vol}
 ]
generalBaseFrustum = frustum[h, rbig, β, "baseangle"]
test @ depthFromVolume[generalBaseFrustum, vol];
```

frustum[h, rbig, β, baseangle]

depthFromVolume[generalBaseFrustum, vol] → $\left(rbig - \left(rbig^3 - \frac{3\, vol \operatorname{Cot}[\beta]}{\pi}\right)^{1/3}\right) \operatorname{Tan}[\beta]$

Height and Depth: Cartesian

```
genericFrustumDepthFromVolumeCartesian[] := Module[{f, ch, fullf, topf, scaledTop, scale, h, rbig, rsmall, vol, a, eqn, solns, soln, depth},
  f = frustum[h, rbig, rsmall];
  fullf = fullCone[f];
  topf = topCone[f];
  scaledTop = scaled[topf, scale];
  a = assumptions[fullf] && assumptions[scaledTop] && vol ≥ 0;
  eqn = (volume[fullf] - volume[scaledTop]) == vol;
  solns = Assuming[a, Solve[eqn, scale]];
  soln = solns[[2]];
  depth = FullSimplify[(height[fullf] - height[scaledTop]) /. soln , a];
  genericFrustumDepthFromVolumeCartesian[] = { h, rbig, rsmall, vol, depth }
 ]
test @ genericFrustumDepthFromVolumeCartesian[];
```

genericFrustumDepthFromVolumeCartesian[] →

$$\left\{h\$10028,\ rbig\$10028,\ rsmall\$10028,\ vol\$10028,\ \frac{h\$10028\ rbig\$10028 - h\$10028^{2/3}\left(h\$10028\ rbig\$10028^3 + \frac{3\,(-rbig\$10028+rsmall\$10028)\ vol\$10028}{\pi}\right)^{1/3}}{rbig\$10028 - rsmall\$10028}\right\}$$

We compute depth from volume two different ways, then show they’re the same. We then choose for use the version that avoids trigonometry (in the apex-angled conversion).

```
depthFromVolume1[f : frustum[ignored_, rbig_, rsmall_], vol_] := Module[{hh, rr, αα, vv, eqn, depth},
  {hh, rr, αα, vv, depth} = genericFrustumDepthFromVolumeApex[];
  depth /. {rr → rbig, αα → apexangle[f], vv → vol}
 ]
depthFromVolume2[f : frustum[h_, rbig_, rsmall_], vol_] := Module[{hh, rrbig, rrsmall, vv, eqn, depth},
  { hh, rrbig, rrsmall, vv, depth } = genericFrustumDepthFromVolumeCartesian[];
  depth /. {hh → h, rrbig → rbig, rrsmall → rsmall, vv → vol }
 ]
generalFrustum = frustum[h, rbig, rsmall]
test @ depthFromVolume1[generalFrustum, vol];
test @ depthFromVolume2[generalFrustum, vol];
Module[{d = (rbig - rsmall), r1 = %%, r2 = %, fn, rules},
 rules = {rbig^3 → t1, (rbig - rsmall) → t2, (-rbig + rsmall) → -t2, -3 t2 vol / Pi → t3};
 fn = Function[r, (((Expand[-r * d] + h rbig) //. rules) ) ^3];
 fn[r1] / fn[r2] // FullSimplify
]
depthFromVolume[f : frustum[h_, rbig_, rsmall_], vol_] := depthFromVolume2[f, vol]
```

frustum[h, rbig, rsmall]

depthFromVolume1[generalFrustum, vol] → $\frac{h\left(rbig - \left(rbig^3 - \frac{3\,(rbig-rsmall)\ vol}{h\,\pi}\right)^{1/3}\right)}{rbig - rsmall}$

depthFromVolume2[generalFrustum, vol] → $\frac{h\ rbig - h^{2/3}\left(h\ rbig^3 + \frac{3\,(-rbig+rsmall)\ vol}{\pi}\right)^{1/3}}{rbig - rsmall}$

1

Volume from Depth

```
volumeFromDepth[f : frustum[h_, rbig_, α_, "apexangle"], depth_] :=
 genericVolumeFromDepthUsingInverse[frustum[hh, rrBig, αα, "apexangle"], dd] /. {hh → h, rrBig → rbig, αα → α, dd → depth}
volumeFromDepth[f : frustum[h_, rbig_, β_, "baseangle"], depth_] :=
 genericVolumeFromDepthUsingInverse[frustum[hh, rrBig, ββ, "baseangle"], dd] /. {hh → h, rrBig → rbig, ββ → β, dd → depth}
volumeFromDepth[f : frustum[h_, rbig_, rsmall_], depth_] :=
 genericVolumeFromDepthUsingInverse[frustum[hh, rrBig, rrSmall], dd] /. {hh → h, rrBig → rbig, rrSmall → rsmall, dd → depth}

test @ volumeFromDepth[frustum[h, rbig, α, "apexangle"], depth];
test @ volumeFromDepth[frustum[h, rbig, β, "baseangle"], depth];
test @ volumeFromDepth[frustum[h, rbig, rsmall], depth];
```

volumeFromDepth[frustum[h, rbig, α, apexangle], depth] → $\frac{1}{3}\text{depth}\,\pi\left(3\,\text{rbig}^2+\text{depth}\,\text{Tan}[\alpha]\,(-3\,\text{rbig}+\text{depth}\,\text{Tan}[\alpha])\right)$

volumeFromDepth[frustum[h, rbig, β, baseangle], depth] → $\frac{1}{3}\text{depth}\,\pi\left(3\,\text{rbig}^2+\text{depth}\,\text{Cot}[\beta]\,(-3\,\text{rbig}+\text{depth}\,\text{Cot}[\beta])\right)$

volumeFromDepth[frustum[h, rbig, rsmall], depth] → $\frac{\text{depth}\,\pi\left(3\,h^2\,\text{rbig}^2+\text{depth}^2\,(\text{rbig}-\text{rsmall})^2+3\,\text{depth}\,h\,\text{rbig}\,(-\text{rbig}+\text{rsmall})\right)}{3\,h^2}$

Radius from Depth

```
radiusFromDepth[f : frustum[h_, rbig_, β_, "baseangle"], depth_] := Block[{eqn, result},
   (*eqn = depth / (rbig - result) == Tan[β];
   result /. First @ Solve[eqn, result]*)
   rbig - depth Cot[β]]
radiusFromDepth[f : frustum[h_, rbig_, α_, "apexangle"], depth_] := radiusFromDepth[toBaseAngled[f], depth]
radiusFromDepth[f : frustum[h_, rbig_, rsmall_], depth_] := FullSimplify[radiusFromDepth[toBaseAngled[f], depth], assumptions[f]]

test @ radiusFromDepth[frustum[h, rbig, α, "apexangle"], depth];
test @ radiusFromDepth[frustum[h, rbig, β, "baseangle"], depth];
test @ radiusFromDepth[frustum[h, rbig, rsmall], depth];
```

radiusFromDepth[frustum[h, rbig, α, apexangle], depth] → $\text{rbig}-\text{depth}\,\text{Tan}[\alpha]$

radiusFromDepth[frustum[h, rbig, β, baseangle], depth] → $\text{rbig}-\text{depth}\,\text{Cot}[\beta]$

radiusFromDepth[frustum[h, rbig, rsmall], depth] → $\text{rbig}+\frac{\text{depth}\,(-\text{rbig}+\text{rsmall})}{h}$

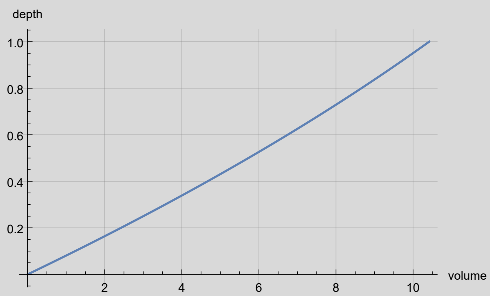
Testing

```
example = frustum[1, 2, π/9, "apexangle"]
{ volume[example], volume[example] // N }
expr = test @ depthFromVolume[example, v];
Plot[expr, {v, 0, volume[example]}, AxesLabel → {"volume", "depth"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ volumeFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "volume"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ radiusFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "radius"}, AxesOrigin → {0, 0}, GridLines → Automatic]
```

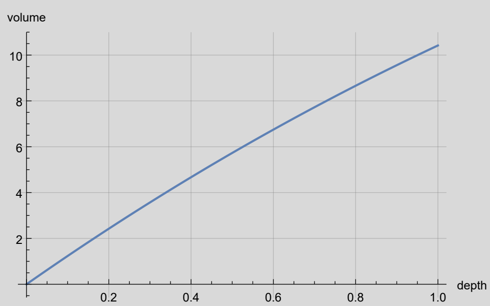
$\text{frustum}\left[1, 2, \frac{\pi}{9}, \text{apexangle}\right]$

$\left\{\frac{1}{3}\pi\left(12+\left(-6+\text{Tan}\left[\frac{\pi}{9}\right]\right)\text{Tan}\left[\frac{\pi}{9}\right]\right), 10.4182\right\}$

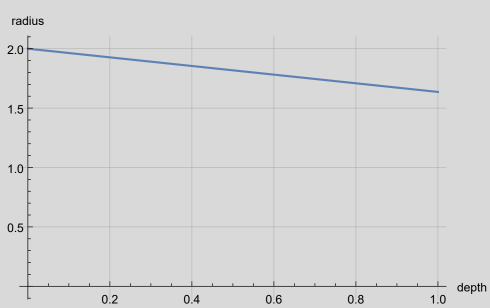
$\text{depthFromVolume}[\text{example}, v] \to \text{Cot}\left[\frac{\pi}{9}\right]\left(2-\left(8-\frac{3\,v\,\text{Tan}\left[\frac{\pi}{9}\right]}{\pi}\right)^{1/3}\right)$



$\text{volumeFromDepth}[\text{example}, \text{depth}] \to \frac{1}{3}\text{depth}\,\pi\left(12+\text{depth}\,\text{Tan}\left[\frac{\pi}{9}\right]\left(-6+\text{depth}\,\text{Tan}\left[\frac{\pi}{9}\right]\right)\right)$



$\text{radiusFromDepth}[\text{example}, \text{depth}] \to 2-\text{depth}\,\text{Tan}\left[\frac{\pi}{9}\right]$

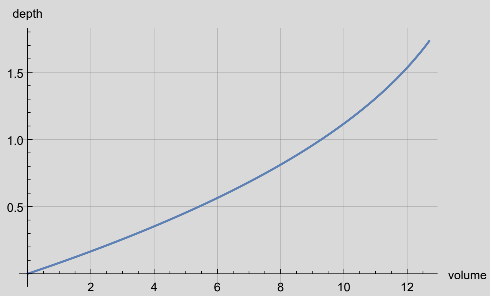


```
example = frustum[Sqrt[3], 2, 1]
{ volume[example], volume[example] // N }
expr = test @ depthFromVolume[example, v];
Plot[expr, {v, 0, volume[example]}, AxesLabel → {"volume", "depth"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ volumeFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "volume"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ radiusFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "radius"}, AxesOrigin → {0, 0}, GridLines → Automatic]
```

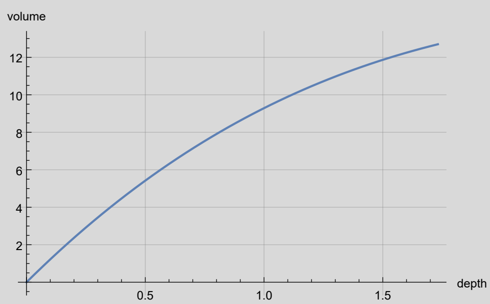
frustum$\left[\sqrt{3}, 2, 1\right]$

$\left\{\frac{7\pi}{\sqrt{3}}, 12.6966\right\}$

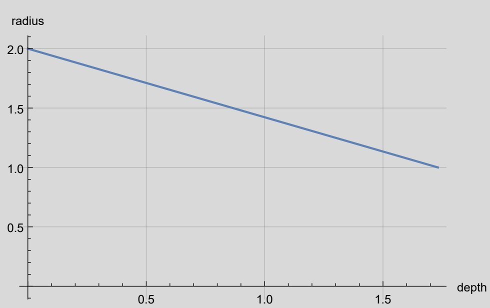
depthFromVolume[example, v] $\to 2\sqrt{3} - 3^{1/3}\left(8\sqrt{3} - \frac{3\,v}{\pi}\right)^{1/3}$



volumeFromDepth[example, depth] $\to \frac{1}{9}\,\text{depth}\left(36 - 6\sqrt{3}\,\text{depth} + \text{depth}^2\right)\pi$



radiusFromDepth[example, depth] $\to 2 - \frac{\text{depth}}{\sqrt{3}}$

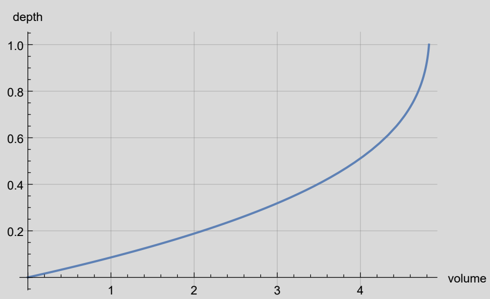


```
example = frustum[1, 2, π / 6, "baseangle"]
{ volume[example], volume[example] // N }
expr = test @ depthFromVolume[example, v];
Plot[expr, {v, 0, volume[example]}, AxesLabel → {"volume", "depth"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ volumeFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "volume"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ radiusFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "radius"}, AxesOrigin → {0, 0}, GridLines → Automatic]
```

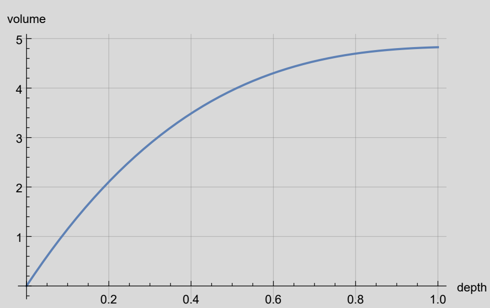
frustum$\left[1, 2, \frac{\pi}{6}, \text{baseangle}\right]$

$\left\{\left(5 - 2\sqrt{3}\right)\pi, 4.82517\right\}$

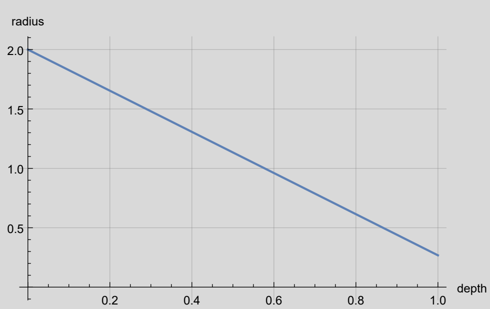
depthFromVolume[example, v] $\rightarrow \dfrac{2 - \left(8 - \frac{3\sqrt{3}\,v}{\pi}\right)^{1/3}}{\sqrt{3}}$



volumeFromDepth[example, depth] $\rightarrow \frac{1}{3}\,\text{depth}\left(12 + \sqrt{3}\,\text{depth}\left(-6 + \sqrt{3}\,\text{depth}\right)\right)\pi$



radiusFromDepth[example, depth] $\rightarrow 2 - \sqrt{3}\,\text{depth}$



Inverted Right Conical Frustum

As we did with cones, we also explore the vertical inversion of the frustum.

Conversion

```
toFrustum[f: invertedFrustum[h_, rbig_, α_, "apexangle"]] := invert @ f
toFrustum[f: invertedFrustum[h_, rbig_, β_, "baseangle"]] := invert @ f
toFrustum[f: invertedFrustum[h_, rbig_, rsmall_]] := invert @ f

invert[f: frustum[h_, rbig_, α_, "apexangle"]] := invertedFrustum[h, rbig, α, "apexangle"]
invert[f: frustum[h_, rbig_, β_, "baseangle"]] := invertedFrustum[h, rbig, β, "baseangle"]
invert[f: frustum[h_, rbig_, rsmall_]] := invertedFrustum[h, rbig, rsmall]

invert[f: invertedFrustum[h_, rbig_, α_, "apexangle"]] := frustum[h, rbig, α, "apexangle"]
invert[f: invertedFrustum[h_, rbig_, β_, "baseangle"]] := frustum[h, rbig, β, "baseangle"]
invert[f: invertedFrustum[h_, rbig_, rsmall_]] := frustum[h, rbig, rsmall]
```

Accessing

```
assumptions[f: invertedFrustum[h_, rbig_, rsmall_]] := assumptions @ toFrustum @ f
assumptions[f: invertedFrustum[h_, rbig_, α_, "apexangle"]] := assumptions @ toFrustum @ f
assumptions[f: invertedFrustum[h_, rbig_, β_, "baseangle"]] := assumptions @ toFrustum @ f
test @ assumptions[invertedFrustum[h, rbig, α, "apexangle"]];
test @ assumptions[invertedFrustum[h, rbig, β, "baseangle"]];
```

```
assumptions[invertedFrustum[h, rbig, α, apexangle]] → h ≥ 0 && rbig ≥ 0 && 2 α < π && α > 0
```

```
assumptions[invertedFrustum[h, rbig, β, baseangle]] → h ≥ 0 && rbig ≥ 0 && β > 0 && 2 β < π
```

```
apexangle[f: invertedFrustum[h_, rbig_, α_, "apexangle"]] := apexangle @ invert @ f
apexangle[f: invertedFrustum[h_, rbig_, β_, "baseangle"]] := apexangle @ invert @ f
apexangle[f: invertedFrustum[h_, rbig_, rsmall_]] := apexangle @ invert @ f

baseangle[f: invertedFrustum[h_, rbig_, α_, "apexangle"]] := baseangle @ invert @ f
baseangle[f: invertedFrustum[h_, rbig_, β_, "baseangle"]] := baseangle @ invert @ f
baseangle[f: invertedFrustum[h_, rbig_, rsmall_]] := baseangle @ invert @ f

baseangle[f: invertedFrustum[h_, rbig_, rbig_ - h_ Cot[β_]]] := baseangle @ invert @ f

test @ apexangle[invertedFrustum[h, rbig, rsmall]];
test @ baseangle[invertedFrustum[h, rbig, rsmall]];
test @ { baseangle[invertedFrustum[1, 3, 2]], baseangle[invertedFrustum[Sqrt[3], 2, 1]]};
```

```
apexangle[invertedFrustum[h, rbig, rsmall]] → ArcTan[h, rbig - rsmall]
```

```
baseangle[invertedFrustum[h, rbig, rsmall]] → ArcTan[rbig - rsmall, h]
```

$\{\text{baseangle}[\text{invertedFrustum}[1, 3, 2]], \text{baseangle}[\text{invertedFrustum}[\sqrt{3}, 2, 1]]\} \to \{\frac{\pi}{4}, \frac{\pi}{3}\}$

```
height[f: invertedFrustum[h_, rbig_, α_, "apexangle"]] := h
height[f: invertedFrustum[h_, rbig_, β_, "baseangle"]] := h
height[f: invertedFrustum[h_, rbig_, rsmall_]] := h
```

```
rbig[f: invertedFrustum[h_, rbig_, α_, "apexangle"]] := rbig
rbig[f: invertedFrustum[h_, rbig_, β_, "baseangle"]] := rbig
rbig[f: invertedFrustum[h_, rbig_, rsmall_]] := rbig
```

```
rsmall[f: invertedFrustum[h_, rbig_, α_, "apexangle"]] := rsmall @ invert @ f
rsmall[f: invertedFrustum[h_, rbig_, β_, "baseangle"]] := rsmall @ invert @ f
rsmall[f: invertedFrustum[h_, rbig_, rsmall_]] := rsmall
rsmall[f: invertedFrustum[h_, rbig_, ArcTan[rbig_ - rsmall_, h_], "baseangle"]] := rsmall
test @ rsmall[invertedFrustum[h, rbig, α, "apexangle"]];
test @ rsmall[invertedFrustum[h, rbig, β, "baseangle"]];
test @ rsmall[invertedFrustum[h, rbig, rsmall]];
```

```
rsmall[invertedFrustum[h, rbig, α, apexangle]] → rbig - h Tan[α]
```

```
rsmall[invertedFrustum[h, rbig, β, baseangle]] → rsmall - h Cot[β]
```

```
rsmall[invertedFrustum[h, rbig, rsmall]] → rsmall
```

Conversion Redux

```
toInvertedFrustum[f : invertedFrustum[h_, rbig_, α_, "apexangle"]] := invertedFrustum[h, rbig, rsmall[f]]
toInvertedFrustum[f : invertedFrustum[h_, rbig_, β_, "baseangle"]] := invertedFrustum[h, rbig, rsmall[f]]
toInvertedFrustum[f : invertedFrustum[h_, rbig_, rsmall_]] := f

toCartesian[f : invertedFrustum[h_, rbig_, α_, "apexangle"]] := toInvertedFrustum @ f
toCartesian[f : invertedFrustum[h_, rbig_, β_, "baseangle"]] := toInvertedFrustum @ f
toCartesian[f : invertedFrustum[h_, rbig_, rsmall_]] := toInvertedFrustum @ f

toApexAngled[f : invertedFrustum[h_, rbig_, α_, "apexangle"]] := f
toApexAngled[f : invertedFrustum[h_, rbig_, β_, "baseangle"]] := invert @ toApexAngled @ invert @ f
toApexAngled[f : invertedFrustum[h_, rbig_, rsmall_]] := invert @ toApexAngled @ invert @ f

toBaseAngled[f : invertedFrustum[h_, rbig_, α_, "apexangle"]] := invert @ toBaseAngled @ invert @ f
toBaseAngled[f : invertedFrustum[h_, rbig_, β_, "baseangle"]] := f
toBaseAngled[f : invertedFrustum[h_, rbig_, rsmall_]] := invert @ toBaseAngled @ invert @ f
```

```
test @ toCartesian @ invertedFrustum[h, rbig, β, "baseangle"];
test @ toBaseAngled @ %;
test @ toApexAngled @ %%;
test @ toFrustum @ %;
test @ toBaseAngled @ %%;
```

toCartesian[invertedFrustum[h, rbig, β, baseangle]] → invertedFrustum[h, rbig, rsmall - h Cot[β]]

toBaseAngled[%] → invertedFrustum[h, rbig, ArcTan[rbig - rsmall + h Cot[β], h], baseangle]

toApexAngled[%%] → invertedFrustum[h, rbig, ArcTan[h, rbig - rsmall + h Cot[β]], apexangle]

toFrustum[%] → frustum[h, rbig, ArcTan[h, rbig - rsmall + h Cot[β]], apexangle]

toBaseAngled[%%] → invertedFrustum[h, rbig, $\frac{\pi}{2}$ - ArcTan[h, rbig - rsmall + h Cot[β]], baseangle]

```
test @ toBaseAngled @ invertedFrustum[h, rbig, rsmall];
test @ toCartesian @ %;
```

toBaseAngled[invertedFrustum[h, rbig, rsmall]] → invertedFrustum[h, rbig, ArcTan[rbig - rsmall, h], baseangle]

toCartesian[%] → invertedFrustum[h, rbig, rsmall]

Volume

```
coneHeight[f:invertedFrustum[h_, rbig_, α_, "apexangle"]] := coneHeight @ invert @ f
coneHeight[f:invertedFrustum[h_, rbig_, β_, "baseangle"]] := coneHeight @ invert @ f
coneHeight[f:invertedFrustum[h_, rbig_, rsmall_]] := coneHeight @ invert @ f

test @ coneHeight[invertedFrustum[h, rbig, α, "apexangle"]];
test @ coneHeight[invertedFrustum[h, rbig, β, "baseangle"]];
test @ toApexAngled @ invertedFrustum[h, rbig, β, "baseangle"];
test @ coneHeight @ %;
test @ coneHeight[invertedFrustum[h, rbig, rsmall]];
test @ coneHeight[invertedFrustum[1, 3, 2]];
```

coneHeight[invertedFrustum[h, rbig, α, apexangle]] → rbig Cot[α]

coneHeight[invertedFrustum[h, rbig, β, baseangle]] → rbig Tan[β]

toApexAngled[invertedFrustum[h, rbig, β, baseangle]] → invertedFrustum$\left[\text{h, rbig, } \frac{\pi}{2} - \beta\text{, apexangle}\right]$

coneHeight[%] → rbig Tan[β]

coneHeight[invertedFrustum[h, rbig, rsmall]] → $\frac{\text{h rbig}}{\text{rbig} - \text{rsmall}}$

coneHeight[invertedFrustum[1, 3, 2]] → 3

```
volume[f: invertedFrustum[h_, rbig_, rsmall_]] := volume @ invert @ f
volume[f: invertedFrustum[h_, rbig_, α_, "apexangle"]] := volume @ invert @ f
volume[f: invertedFrustum[h_, rbig_, β_, "baseangle"]] := volume @ invert @ f

v = test @ volume[invertedFrustum[h, r1, r2]]; (* compare to textbook answer 1/3 h π (r1²+r1 r2+r2²) *)
vα = test @ volume[invertedFrustum[h, r, α, "apexangle"]];
test @ toCartesian @ invertedFrustum[h, r, α, "apexangle"];
vα2 = test @ volume[%];
vβ = test @ volume[invertedFrustum[h, r, β, "baseangle"]];
test @ (v /. r2 → 0);
Clear[v, vα, vα2, vβ]
```

volume[invertedFrustum[h, r1, r2]] → $\frac{1}{3} h \pi \left(r1^2 + r1\, r2 + r2^2\right)$

volume[invertedFrustum[h, r, α, apexangle]] → $\frac{1}{3} h \pi \left(3 r^2 + h \operatorname{Tan}[\alpha] \left(-3 r + h \operatorname{Tan}[\alpha]\right)\right)$

toCartesian[invertedFrustum[h, r, α, apexangle]] → invertedFrustum[h, r, r − h Tan[α]]

volume[%] → $\frac{1}{3} \pi \operatorname{Cot}[\alpha] \left(r^3 - (r - h \operatorname{Tan}[\alpha])^3\right)$

volume[invertedFrustum[h, r, β, baseangle]] → $\frac{1}{3} h \pi \left(3 r^2 + h \operatorname{Cot}[\beta] \left(-3 r + h \operatorname{Cot}[\beta]\right)\right)$

(v /. r2 → 0) → $\frac{1}{3} h \pi\, r1^2$

Height and Depth

We’re looking for a frustum with same base angle and bottom radius, but different height

```
depthFromVolume[f:invertedFrustum[h_, rbig_, α_, "apexangle"], vol_] := Module[{},
  h - depthFromVolume[invert @ f, volume[f] - vol] // Simplify
 ]
generalApexInvertedFrustum = invertedFrustum[h, r, α, "apexangle"]
test @ depthFromVolume[generalApexInvertedFrustum, vol];
```

invertedFrustum[h, r, α, apexangle]

depthFromVolume[generalApexInvertedFrustum, vol] $\to h + \text{Cot}[\alpha]\left(-r + \left(r^3 - 3\,h\,r^2\,\text{Tan}[\alpha] + \frac{3\,\text{vol}\,\text{Tan}[\alpha]}{\pi} + 3\,h^2\,r\,\text{Tan}[\alpha]^2 - h^3\,\text{Tan}[\alpha]^3\right)^{1/3}\right)$

```
depthFromVolume[f:invertedFrustum[h_, rbig_, rsmall_], vol_] := Module[{},
  h - depthFromVolume[invert @ f, volume[f] - vol] // FullSimplify
 ]
generalInvertedFrustum = invertedFrustum[h, rbig, rsmall]
test @ depthFromVolume[generalInvertedFrustum, vol];
```

invertedFrustum[h, rbig, rsmall]

depthFromVolume[generalInvertedFrustum, vol] $\to \frac{h\,\text{rsmall} - h^{2/3}\left(h\,\text{rsmall}^3 + \frac{3\,(\text{rbig}-\text{rsmall})\,\text{vol}}{\pi}\right)^{1/3}}{-\text{rbig} + \text{rsmall}}$

```
depthFromVolume[f:invertedFrustum[h_, rbig_, β_, "baseangle"], vol_] := Module[{hh, rr, αα, vv, eqn, soln},
  h - depthFromVolume[invert @ f, volume[f] - vol] // FullSimplify
 ]
generalBaseInvertedFrustum = invertedFrustum[h, r, β, "baseangle"]
test @ depthFromVolume[generalBaseInvertedFrustum, vol];
```

invertedFrustum[h, r, β, baseangle]

depthFromVolume[generalBaseInvertedFrustum, vol] $\to h + \left(-r + \left(r^3 + \frac{\text{Cot}[\beta]\left(-3\,h\,\pi\,r^2 + 3\,\text{vol} + h^2\,\pi\,\text{Cot}[\beta]\,(3\,r - h\,\text{Cot}[\beta])\right)}{\pi}\right)^{1/3}\right)\text{Tan}[\beta]$

```
volumeFromDepth0[f:invertedFrustum[h_, rbig_, rsmall_], depth_] :=
 FullSimplify[volume[f] - volumeFromDepth[invert[f], h - depth], assumptions[f]]
volumeFromDepth0[f:invertedFrustum[h_, rbig_, α_, "apexangle"], depth_] := volumeFromDepth[toCartesian[f], depth]
volumeFromDepth0[f:invertedFrustum[h_, rbig_, β_, "baseangle"], depth_] :=
 FullSimplify[volume[f] - volumeFromDepth[invert[f], h - depth], assumptions[f]]

test @ volumeFromDepth0[invertedFrustum[h, rbig, α, "apexangle"], depth];
test @ volumeFromDepth0[invertedFrustum[h, rbig, β, "baseangle"], depth];
test @ volumeFromDepth0[invertedFrustum[h, rbig, rsmall], depth];
```

volumeFromDepth0[invertedFrustum[h, rbig, α, apexangle], depth] → volumeFromDepth[invertedFrustum[h, rbig, rbig − h Tan[α]], depth]

volumeFromDepth0[invertedFrustum[h, rbig, β, baseangle], depth] $\to \frac{1}{3}\,\text{depth}\,\pi\left(3\,\text{rbig}^2 + 3\,(\text{depth} - 2\,h)\,\text{rbig}\,\text{Cot}[\beta] + \left(\text{depth}^2 - 3\,\text{depth}\,h + 3\,h^2\right)\text{Cot}[\beta]^2\right)$

volumeFromDepth0[invertedFrustum[h, rbig, rsmall], depth] $\to \frac{\text{depth}\,\pi\left(\text{depth}^2\,(\text{rbig} - \text{rsmall})^2 + 3\,\text{depth}\,h\,(\text{rbig} - \text{rsmall})\,\text{rsmall} + 3\,h^2\,\text{rsmall}^2\right)}{3\,h^2}$

```
volumeFromDepth[f : invertedFrustum[h_, rbig_, α_, "apexangle"], depth_] :=
 genericVolumeFromDepthUsingInverse[invertedFrustum[hh, rrBig, αα, "apexangle"], dd] /. {hh → h, rrBig → rbig, αα → α, dd → depth}
volumeFromDepth[f : invertedFrustum[h_, rbig_, β_, "baseangle"], depth_] :=
 genericVolumeFromDepthUsingInverse[invertedFrustum[hh, rrBig, ββ, "baseangle"], dd] /. {hh → h, rrBig → rbig, ββ → β, dd → depth}
volumeFromDepth[f : invertedFrustum[h_, rbig_, rsmall_], depth_] :=
 genericVolumeFromDepthUsingInverse[invertedFrustum[hh, rrBig, rrSmall], dd] /. {hh → h, rrBig → rbig, rrSmall → rsmall, dd → depth}

test @ volumeFromDepth[invertedFrustum[h, rbig, α, "apexangle"], depth];
test @ volumeFromDepth[invertedFrustum[h, rbig, β, "baseangle"], depth];
test @ volumeFromDepth[invertedFrustum[h, rbig, rsmall], depth];
```

volumeFromDepth[invertedFrustum[h, rbig, α, apexangle], depth] $\rightarrow \frac{1}{3}\,\text{depth}\,\pi\left(3\,\text{rbig}^2+3\,(\text{depth}-2\,\text{h})\,\text{rbig}\,\text{Tan}[\alpha]+\left(\text{depth}^2-3\,\text{depth}\,\text{h}+3\,\text{h}^2\right)\text{Tan}[\alpha]^2\right)$

volumeFromDepth[invertedFrustum[h, rbig, β, baseangle], depth] $\rightarrow \frac{1}{3}\,\text{depth}\,\pi\left(3\,\text{rbig}^2+3\,(\text{depth}-2\,\text{h})\,\text{rbig}\,\text{Cot}[\beta]+\left(\text{depth}^2-3\,\text{depth}\,\text{h}+3\,\text{h}^2\right)\text{Cot}[\beta]^2\right)$

volumeFromDepth[invertedFrustum[h, rbig, rsmall], depth] $\rightarrow \frac{\text{depth}\,\pi\left(\text{depth}^2\,(\text{rbig}-\text{rsmall})^2+3\,\text{depth}\,\text{h}\,(\text{rbig}-\text{rsmall})\,\text{rsmall}+3\,\text{h}^2\,\text{rsmall}^2\right)}{3\,\text{h}^2}$

```
radiusFromDepth[f : invertedFrustum[h_, rbig_, β_, "baseangle"], depth_] := Block[{eqn, result},
  (*eqn = (h - depth) / (rbig - result) == Tan[β];
  Simplify[result /. First @ Solve[eqn, result], assumptions[f]]*)
  rbig + (depth - h) Cot[β]]
radiusFromDepth[f : invertedFrustum[h_, rbig_, α_, "apexangle"], depth_] := radiusFromDepth[toBaseAngled[f], depth]
radiusFromDepth[f : invertedFrustum[h_, rbig_, rsmall_], depth_] := radiusFromDepth[toBaseAngled[f], depth]

test @ radiusFromDepth[invertedFrustum[h, rbig, α, "apexangle"], depth];
test @ radiusFromDepth[invertedFrustum[h, rbig, β, "baseangle"], depth];
test @ radiusFromDepth[invertedFrustum[h, rbig, rsmall], depth];
```

radiusFromDepth[invertedFrustum[h, rbig, α, apexangle], depth] $\rightarrow$ rbig + (depth − h) Tan[α]

radiusFromDepth[invertedFrustum[h, rbig, β, baseangle], depth] $\rightarrow$ rbig + (depth − h) Cot[β]

radiusFromDepth[invertedFrustum[h, rbig, rsmall], depth] $\rightarrow \text{rbig}+\frac{(\text{depth}-\text{h})\,(\text{rbig}-\text{rsmall})}{\text{h}}$

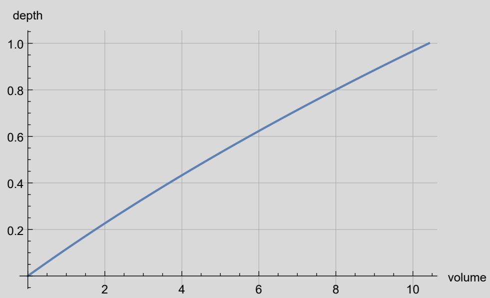
Testing

```
example = invertedFrustum[1, 2, π/9, "apexangle"]
{ volume[example], volume[example] // N }
expr = test @ depthFromVolume[example, v];
Plot[expr, {v, 0, volume[example]}, AxesLabel → {"volume", "depth"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ volumeFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "volume"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ radiusFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "radius"}, AxesOrigin → {0, 0}, GridLines → Automatic]
```

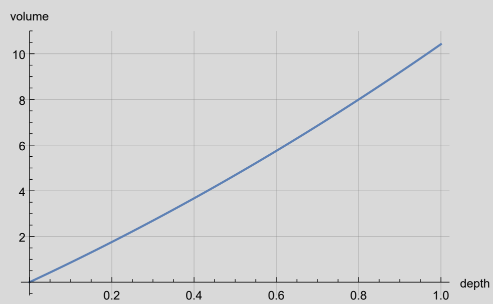
$\text{invertedFrustum}\left[1, 2, \frac{\pi}{9}, \text{apexangle}\right]$

$\left\{\frac{1}{3}\pi\left(12+\left(-6+\text{Tan}\left[\frac{\pi}{9}\right]\right)\text{Tan}\left[\frac{\pi}{9}\right]\right), 10.4182\right\}$

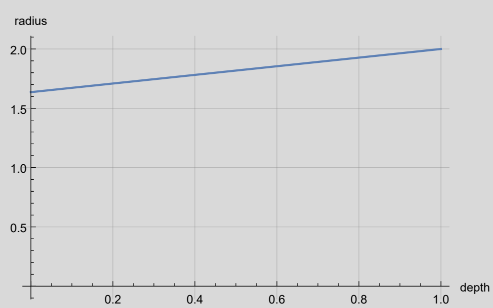
$\text{depthFromVolume}[\text{example}, v] \to 1-2\,\text{Cot}\left[\frac{\pi}{9}\right]+\frac{\left(3\,v\,\text{Cot}\left[\frac{\pi}{9}\right]^2+\pi\left(-1+2\,\text{Cot}\left[\frac{\pi}{9}\right]\right)^3\right)^{1/3}}{\pi^{1/3}}$



$\text{volumeFromDepth}[\text{example}, \text{depth}] \to \frac{1}{3}\,\text{depth}\,\pi\left(12+6\,(-2+\text{depth})\,\text{Tan}\left[\frac{\pi}{9}\right]+\left(3-3\,\text{depth}+\text{depth}^2\right)\text{Tan}\left[\frac{\pi}{9}\right]^2\right)$



$\text{radiusFromDepth}[\text{example}, \text{depth}] \to 2+(-1+\text{depth})\,\text{Tan}\left[\frac{\pi}{9}\right]$

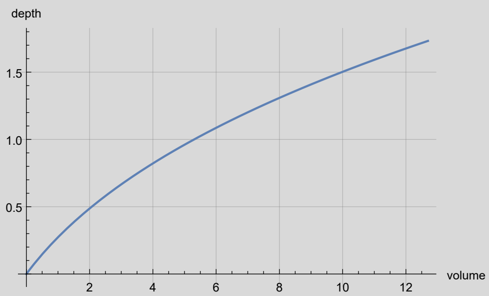


```
example = invertedFrustum[Sqrt[3], 2, 1]
{ volume[example], volume[example] // N }
expr = test @ depthFromVolume[example, v];
Plot[expr, {v, 0, volume[example]}, AxesLabel → {"volume", "depth"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ volumeFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "volume"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ radiusFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "radius"}, AxesOrigin → {0, 0}, GridLines → Automatic]
```

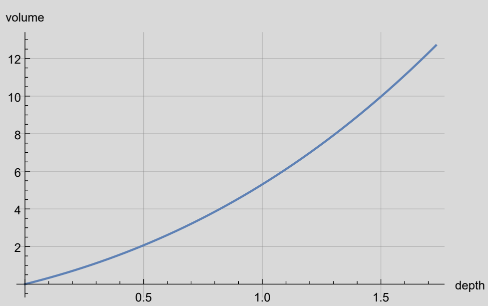
invertedFrustum$\left[\sqrt{3}, 2, 1\right]$

$\left\{\frac{7\pi}{\sqrt{3}}, 12.6966\right\}$

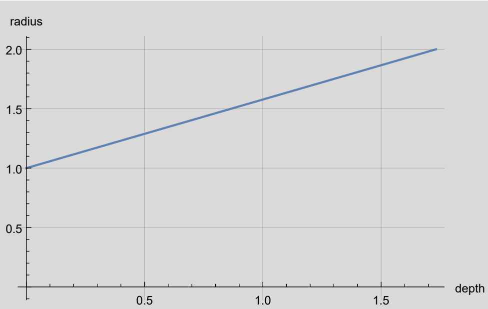
depthFromVolume[example, v] $\to -\sqrt{3} + \left(3\sqrt{3} + \frac{9\,v}{\pi}\right)^{1/3}$



volumeFromDepth[example, depth] $\to \frac{1}{9}\,\text{depth}\left(9 + 3\sqrt{3}\,\text{depth} + \text{depth}^2\right)\pi$



radiusFromDepth[example, depth] $\to 2 + \frac{-\sqrt{3} + \text{depth}}{\sqrt{3}}$

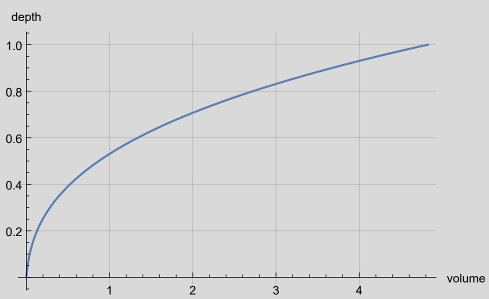


```
example = invertedFrustum[1, 2, π / 6, "baseangle"]
{ volume[example], volume[example] // N }
expr = test @ depthFromVolume[example, v];
Plot[expr, {v, 0, volume[example]}, AxesLabel → {"volume", "depth"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ volumeFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "volume"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ radiusFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "radius"}, AxesOrigin → {0, 0}, GridLines → Automatic]
```

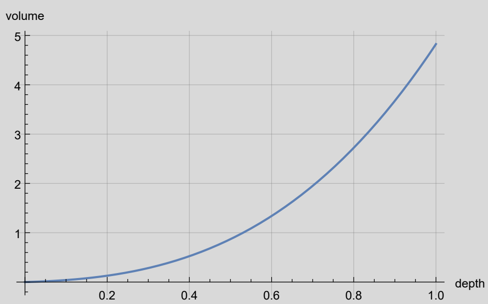
$\text{invertedFrustum}\left[1, 2, \frac{\pi}{6}, \text{baseangle}\right]$

$\left\{\left(5-2\sqrt{3}\right)\pi, 4.82517\right\}$

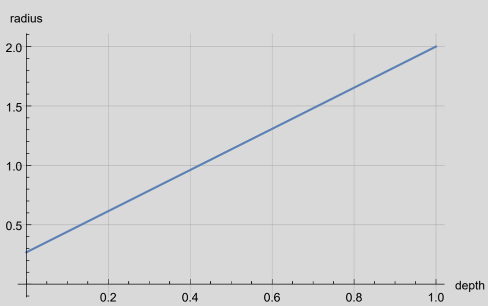
$\text{depthFromVolume}[\text{example}, v] \to 1-\frac{2}{\sqrt{3}}+\frac{\left(26-15\sqrt{3}+\frac{3\sqrt{3}\,v}{\pi}\right)^{1/3}}{\sqrt{3}}$



$\text{volumeFromDepth}[\text{example}, \text{depth}] \to \frac{1}{3}\,\text{depth}\left(12+6\sqrt{3}\,(-2+\text{depth})+3\left(3-3\,\text{depth}+\text{depth}^2\right)\right)\pi$



$\text{radiusFromDepth}[\text{example}, \text{depth}] \to 2+\sqrt{3}\,(-1+\text{depth})$



Sphere

We don’t do much with spheres, but have a little bit of logic, just in case.

Accessing

```
assumptions[sphere[r_]] := r ≥ 0
radius[sphere[r_]] := r
```

Volume

```
volume[sphere[r_]] := Module[{α},
  4 / 3 Pi r^3
 ]
test @ volume[sphere[r]];
```

volume[sphere[r]] → $\frac{4 \pi r^3}{3}$

Inverted Spherical Cap (i.e.: a Bowl)

See http://mathworld.wolfram.com/SphericalCap.html. And (more usefully) https://en.wikipedia.org/wiki/Spherical_cap. By ‘inverted’ spherical cap, we here mean a cap on the bottom of the sphere instead of the top. Think of a bowl.

Accessing

```
rCap[c:invertedSphericalCap[h_, rSphere_, "rSphere"]] := Sqrt[rSphere^2 - (rSphere-h)^2]
rSphere[c:invertedSphericalCap[h_, rSphere_, "rSphere"]] := rSphere
height[c:invertedSphericalCap[h_, rSphere_, "rSphere"]] := h

rCap[c:invertedSphericalCap[h_, a_, "rCap"]] := a
rSphere[c:invertedSphericalCap[h_, a_, "rCap"]] := (a^2 + h^2) / (2 h)
height[c:invertedSphericalCap[h_, a_, "rCap"]] := h

assumptions[c: invertedSphericalCap[h_, rSphere_, "rSphere"]] := rSphere > 0 && h > 0 && rSphere ≥ h
assumptions[c: invertedSphericalCap[h_, a_, "rCap"]] := h > 0 && a > 0 && rSphere[c] ≥ h
```

Conversion

```
toCap[c:invertedSphericalCap[h_, rSphere_, "rSphere"]] := invertedSphericalCap[h, rCap[c], "rCap"]
toCap[c:invertedSphericalCap[h_, a_, "rCap"]] := c
toSphere[c:invertedSphericalCap[h_, a_, "rCap"]] := invertedSphericalCap[h, rSphere[c], "rSphere"]
toSphere[c:invertedSphericalCap[h_, rSphere_, "rSphere"]] := c

toCartesian[c:invertedSphericalCap[h_, rSphere_, "rSphere"]] := c
toCartesian[c:invertedSphericalCap[h_, a_, "rCap"]] := c
```

Volume

Formulas from Wikipedia

```
volume[invertedSphericalCap[h_, rSphere_, "rSphere"]] := Block[{},
  π / 3 * h^2 * (3 rSphere - h)
 ]
volume[invertedSphericalCap[h_, a_, "rCap"]] := Block[{},
  1 / 6 * π * h * (3 a^2 + h^2)
 ]
test@volume[invertedSphericalCap[h, r, "rSphere"]];
test@volume[invertedSphericalCap[h, a, "rCap"]];
```

volume[invertedSphericalCap[h, r, rSphere]] → $\frac{1}{3} h^2 \pi (-h + 3 r)$

volume[invertedSphericalCap[h, a, rCap]] → $\frac{1}{6} h \left(3 a^2 + h^2\right) \pi$

Height and Depth

```
Clear[genericSphericalCapDepthFromVolume]
genericSphericalCapDepthFromVolume[] := Module[{cap, a, h, vol, assumpts, eqn, solns, soln, c1, break},
   cap = invertedSphericalCap[h, a, "rCap"];
   assumpts = assumptions[cap] && vol ≥ 0;
   eqn = vol == volume[cap];
   solns = Assuming[assumpts, Solve[eqn, h]];
   soln = h /. solns[[1]];
   genericSphericalCapDepthFromVolume[] = {h, a, vol, soln}
  ];
```

```
depthFromVolume[c : invertedSphericalCap[h_, r_, "rSphere"], v_] := depthFromVolume[toCap[c], v]
depthFromVolume[c : invertedSphericalCap[h_, a_, "rCap"], v_] := Module[{aa, hh, vol, soln},
  {hh, aa, vol, soln} = genericSphericalCapDepthFromVolume[];
  (soln /. {aa → a, hh → h, vol → v})
 ]
test @ depthFromVolume[invertedSphericalCap[1, 2, "rCap"], volume[invertedSphericalCap[1, 2, "rCap"]]];
N @ %
test @ depthFromVolume[invertedSphericalCap[h, r, "rCap"], volume];
N @ %
```

depthFromVolume[invertedSphericalCap[1, 2, rCap], volume[invertedSphericalCap[1, 2, rCap]]] $\to 4\left(\frac{\pi}{-\frac{13\pi}{2}+\frac{5\sqrt{17}\,\pi}{2}}\right)^{1/3}-\left(\frac{-\frac{13\pi}{2}+\frac{5\sqrt{17}\,\pi}{2}}{\pi}\right)^{1/3}$

1.

depthFromVolume[invertedSphericalCap[h, r, rCap], volume] $\to \frac{\pi^{1/3}\,r^2}{\left(-3\,\text{volume}+\sqrt{\pi^2 r^6+9\,\text{volume}^2}\right)^{1/3}}-\frac{\left(-3\,\text{volume}+\sqrt{\pi^2 r^6+9\,\text{volume}^2}\right)^{1/3}}{\pi^{1/3}}$

$$\frac{1.46459\,r^2}{\left(-3.\,\text{volume}+\sqrt{9.8696\,r^6+9.\,\text{volume}^2}\right)^{1/3}}-0.682784\left(-3.\,\text{volume}+\sqrt{9.8696\,r^6+9.\,\text{volume}^2}\right)^{1/3}$$

```
volumeFromDepth[c : invertedSphericalCap[h_, r_, "rSphere"], depth_] := volumeFromDepth[toCap[c], depth]
volumeFromDepth[c : invertedSphericalCap[h_, a_, "rCap"], depth_] := FullSimplify[
  genericVolumeFromDepthUsingInverse[invertedSphericalCap[ hh, aa, "rCap"], dd] /. {aa → a, hh → h, dd → depth}, assumptions[c] && depth > 0]

test @ volumeFromDepth[invertedSphericalCap[h, a, "rCap"], depth];
test @ volumeFromDepth[invertedSphericalCap[h, r, "rSphere"], depth];
```

volumeFromDepth[invertedSphericalCap[h, a, rCap], depth] $\to \frac{1}{6}\left(3\,a^2\,\text{depth}+\text{depth}^3\right)\pi$

volumeFromDepth[invertedSphericalCap[h, r, rSphere], depth] $\to \frac{1}{6}\pi\left(\text{depth}^3-3\,\text{depth}\,h\,(h-2\,r)\right)$

For radiusFromDepth, we refer to the radius of top of the portion of the cap that is occupied for a given depth.

```
Clear[genericSphericalCapRadiusFromDepth]
genericSphericalCapRadiusFromDepth[] := Module[{c, a, h, r, depth, result, assumpts, eqn, solns, soln, c1, break},
   c = invertedSphericalCap[h, a, "rCap"];
   assumpts = assumptions[c] && depth ≥ 0;
   r = rSphere[c];
   eqn = (r - depth) ^2 + result^2 == r^2;
   soln = FullSimplify[result /. Solve[eqn, result][[1]], assumptions[c]]; (* can take either soln, as we square and then Sqrt *)
   soln = FullSimplify[Sqrt[soln ^2], assumptions[c]];
   genericSphericalCapRadiusFromDepth[] = {h, a , depth, soln}
  ];
```

```
radiusFromDepth[c : invertedSphericalCap[h_, r_, "rSphere"], depth_] := radiusFromDepth[toCap[c], depth]
radiusFromDepth[c : invertedSphericalCap[h_, a_, "rCap"], depth_] := Module[{aa, hh, dd, soln},
  {hh, aa, dd, soln} = genericSphericalCapRadiusFromDepth[];
  FullSimplify[(soln /. {aa → a, hh → h, dd → depth}), assumptions[c]]
 ]
test @ radiusFromDepth[invertedSphericalCap[h, a, "rCap"], depth];
test @ radiusFromDepth[invertedSphericalCap[h, r, "rSphere"], depth];

test @ radiusFromDepth[invertedSphericalCap[h, h, "rCap"], h];
test @ radiusFromDepth[invertedSphericalCap[h, h, "rSphere"], h];
```

radiusFromDepth[invertedSphericalCap[h, a, rCap], depth] $\rightarrow \sqrt{\frac{\text{depth}\,\left(a^2 + h\,(-\text{depth} + h)\right)}{h}}$

radiusFromDepth[invertedSphericalCap[h, r, rSphere], depth] $\rightarrow \sqrt{-\text{depth}\,(\text{depth} - 2\,r)}$

radiusFromDepth[invertedSphericalCap[h, h, rCap], h] $\rightarrow$ h

radiusFromDepth[invertedSphericalCap[h, h, rSphere], h] $\rightarrow$ h

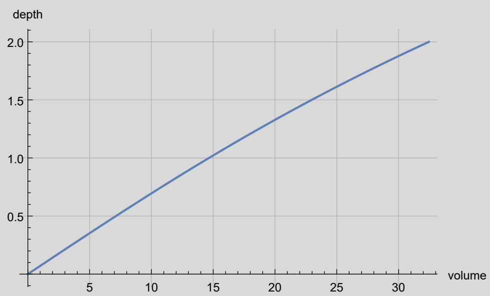
Testing

```
example = invertedSphericalCap[2, 3, "rCap"]
{ volume[example], volume[example] // N }
expr = test @ depthFromVolume[example, v];
Plot[expr, {v, 0, volume[example]}, AxesLabel → {"volume", "depth"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ volumeFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "volume"}, AxesOrigin → {0, 0}, GridLines → Automatic]
expr = test @ radiusFromDepth[example, depth];
Plot[expr, {depth, 0, height[example]}, AxesLabel → {"depth", "radius"}, AxesOrigin → {0, 0}, GridLines → Automatic]
```

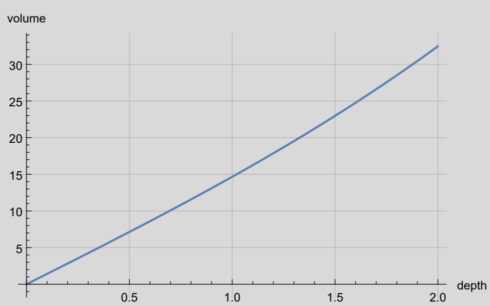
invertedSphericalCap[2, 3, rCap]

$\left\{\frac{31\,\pi}{3}, 32.4631\right\}$

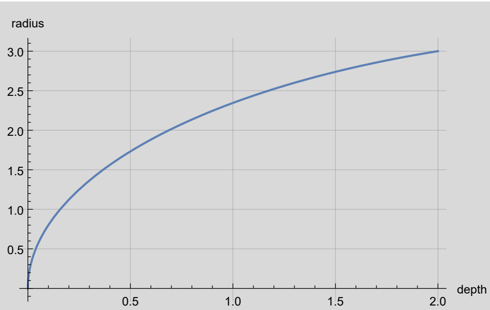
depthFromVolume[example, v] → $\frac{9\,\pi^{1/3}}{\left(-3\,v+\sqrt{729\,\pi^2+9\,v^2}\right)^{1/3}}-\frac{\left(-3\,v+\sqrt{729\,\pi^2+9\,v^2}\right)^{1/3}}{\pi^{1/3}}$



volumeFromDepth[example, depth] → $\frac{1}{6}\,\mathrm{depth}\,\left(27+\mathrm{depth}^2\right)\,\pi$



radiusFromDepth[example, depth] → $\sqrt{\left(\frac{13}{2}-\mathrm{depth}\right)\,\mathrm{depth}}$



Unknown Shape

In unknown shape, where we have uncertainty and currently just give up entirely, we could deduce bounds and use Interval[]. Currently that’s not worthwhile enough to be worth doing.

Note that currently, these are no longer actually used.

```
assumptions[u : unknownShape[h_, vol_]] := h ≥ 0 && vol ≥ 0
test @ assumptions[unknownShape[h, vol]];
```

```
assumptions[unknownShape[h, vol]] → h ≥ 0&&vol ≥ 0
```

```
height[u : unknownShape[h_, vol_]] := h
toCartesian[u : unknownShape[h_, vol_]] := u
```

```
volume[u : unknownShape[h_, vol_]] := Module[{},
  (*printCell[{volume, "h" → h, "vol" → vol}];*)
  vol]
```

```
depthFromVolume[u : unknownShape[h_, vol_], v_] := Piecewise[{
   {0, v ≤ 0 || h ≤ 0 || vol ≤ 0},
   {h, v ≥ vol}
  }, Indeterminate]
```

```
volumeFromDepth[u : unknownShape[h_, vol_], depth_] := Piecewise[{
   {0, depth ≤ 0 || h ≤ 0 || vol ≤ 0},
   {vol, depth ≥ h}
  }, Indeterminate]
```

```
radiusFromDepth[u : unknownShape[h_, vol_], depth_] := Indeterminate
```

```
test @ depthFromVolume[unknownShape[h, vol], v];
test @ volumeFromDepth[unknownShape[h, vol], depth];
```

depthFromVolume[unknownShape[h, vol], v] → $\begin{cases} 0 & v \le 0 \,||\, h \le 0 \,||\, vol \le 0 \\ h & v \ge vol \\ \text{Indeterminate} & \text{True} \end{cases}$

volumeFromDepth[unknownShape[h, vol], depth] → $\begin{cases} 0 & depth \le 0 \,||\, h \le 0 \,||\, vol \le 0 \\ vol & depth \ge h \\ \text{Indeterminate} & \text{True} \end{cases}$

Conical Test Tube

We build up a generic model of a test tube by piecing together simpler building blocks. Generally speaking, we have a mostly-“cylindrical” inverted frustum on top of a “conical” inverted frustum on top of either a flat surface or an inverted spherical cap.

Accessing

```
assumptions[conicalTestTube[cylindrical_, conical_, cap_]] := assumptions[cylindrical] && assumptions[conical] && assumptions[cap]
```

```
toCanonical[c : conicalTestTube[cylindrical_, conical_, cap_]] := c
toCanonical[conicalTestTube[{idTop_, idHip_, idBottom_}, {hTop_, hBottomAndCap_}]] := conicalTestTube[
  (* TODO: use cylinders when we need to *)
  invertedFrustum[hTop, idTop / 2, idHip / 2],
  invertedFrustum[hBottomAndCap - idBottom, idHip / 2, idBottom / 2],
  invertedSphericalCap[idBottom / 2, idBottom / 2, "rCap"]
 ]
```

```
toCartesian[c : conicalTestTube[cylindrical_, conical_, cap_]] := Map[toCartesian, c, {1}]
toApexAngled[c : conicalTestTube[cylindrical_, conical_, cap_]] := Map[toApexAngled, c, {1}]
toBaseAngled[c : conicalTestTube[cylindrical_, conical_, cap_]] := Map[toBaseAngled, c, {1}]
test @ toCartesian[conicalTestTube[cylindrical, conical, cap]];
```

```
toCartesian[conicalTestTube[cylindrical, conical, cap]] → conicalTestTube[toCartesian[cylindrical], toCartesian[conical], toCartesian[cap]]
```

```
height[c : conicalTestTube[cylindrical_, conical_, cap_]] := Total @ (List @@ Map[height, c, {1}])
```

```
parts[c : conicalTestTube[cylindrical_, conical_, cap_]] := {"cylindrical" → cylindrical, "conical" → conical, "cap" → cap} // Association
parts[c : conicalTestTube[idTop_, idHip_, idBottom_, hTop_, hBottom_]] := parts @ toCanonical @ c
test @ parts[toCanonical @ conicalTestTube[{idTop, idHip, idBottom}, {hTop, hBottom}]];
```

parts[toCanonical[conicalTestTube[{idTop, idHip, idBottom}, {hTop, hBottom}]]] → $\langle\left|\text{cylindrical} \to \text{invertedFrustum}\left[\text{hTop}, \frac{\text{idTop}}{2}, \frac{\text{idHip}}{2}\right], \text{conical} \to \text{invertedFrustum}\left[\text{hBottom} - \text{idBottom}, \frac{\text{idHip}}{2}, \frac{\text{idBottom}}{2}\right], \text{cap} \to \text{invertedSphericalCap}\left[\frac{\text{idBottom}}{2}, \frac{\text{idBottom}}{2}, \text{rCap}\right]\right|\rangle$

Volume

```
volume[c : conicalTestTube[cylindrical_, conical_, cap_]] := Total[volume /@ parts[c]]
volume[c : conicalTestTube[idTop_, idHip_, idBottom_, hTop_, hBottom_]] := volume @ toCanonical @ c
```

Height & Depth

```
depthFromVolume[c : conicalTestTube[{idTop_, idHip_, idBottom_}, {hTop_, hBottom_}], v_] := depthFromVolume[toCanonical @ c, v]
depthFromVolume[c : conicalTestTube[cylindrical_, conical_, cap_], v_] :=
 Module[{vCylindrical, vConical, vCap, dFromCap, dFromConical, dOther, result},
  vCap = volume[cap];
  vConical = volume[conical];
  dFromCap = depthFromVolume[cap, v];
  dFromConical = height[cap] + depthFromVolume[conical, v - vCap];
  dOther = height[cap] + height[conical] + depthFromVolume[cylindrical, v - vCap - vConical];
  Piecewise[
   {
    {dFromCap, v ≤ vCap},
    {dFromConical, v ≤ vConical + vCap}, (* had left out the "+ vCap"! *)
    {dOther, True}
   }
  ]
 ]
```

```
volumeFromDepth[c : conicalTestTube[{idTop_, idHip_, idBottom_}, {hTop_, hBottom_}], depth_] := volumeFromDepth[toCanonical @ c, depth]
volumeFromDepth[c : conicalTestTube[cylindrical_, conical_, cap_], depth_] :=
 Module[{hCylindrical, hConical, hCap, vFromCap, vFromConical, vOther, result},
  hCap = height[cap];
  hConical = height[conical];
  vFromCap = volumeFromDepth[cap, depth];
  vFromConical = volume[cap] + volumeFromDepth[conical, depth - hCap];
  vOther = volume[cap] + volume[conical] + volumeFromDepth[cylindrical, depth - hCap - hConical];
  Piecewise[
   {
    {vFromCap, depth ≤ hCap},
    {vFromConical, depth ≤ hConical + hCap},
    {vOther, True}
   }
  ]
 ]
```

```
radiusFromDepth[c : conicalTestTube[{idTop_, idHip_, idBottom_}, {hTop_, hBottom_}], depth_] := radiusFromDepth[toCanonical @ c, depth]
radiusFromDepth[c : conicalTestTube[cylindrical_, conical_, cap_], depth_] :=
 Module[{hCylindrical, hConical, hCap, rFromCap, rFromConical, rOther, result},
  hCap = height[cap];
  hConical = height[conical];
  rFromCap = radiusFromDepth[cap, depth];
  rFromConical = radiusFromDepth[conical, depth - hCap];
  rOther = radiusFromDepth[cylindrical, depth - hCap - hConical];
  Piecewise[
   {
    {rFromCap, depth ≤ hCap},
    {rFromConical, depth ≤ hConical + hCap},
    {rOther, True}
   }
  ]
 ]
```

Pipette and Pipette Tip

Pipettes and tips are defined by their parts from top to bottom, just like the test tubes are.

Accessing

```
assumptions[pipetteTip[parts__]] := And @@ (assumptions /@ {parts})
assumptions[pipette[parts__]] := And @@ (assumptions /@ {parts})
assumptions[mountedPipette[parts__]] := And @@ (assumptions /@ {parts})
test @ assumptions[pipetteTip[invertedFrustum[h2, rbig, rsmall], cone[h1, r]]];
```

```
assumptions[pipetteTip[invertedFrustum[h2, rbig, rsmall], cone[h1, r]]] → h2 ≥ 0&&rbig ≥ 0&&rsmall ≥ 0&&rbig > rsmall&&h1 ≥ 0&&r ≥ 0
```

```
height[pipette[parts__]] := Total[height /@ {parts}]
height[pipetteTip[parts__]] := Total[height /@ {parts}]
height[mountedPipette[parts__]] := Total[height /@ {parts}]
```

Construction

Fancier versions of mountTip would allow for overlap.

Question: should we enforce monotonicity in radius *here?* Would like to, but that sounds hard.

```
Clear[mountTip]
mountTip[p : pipette[pipetteParts__], tip : pipetteTip[tipParts__]] := Module[{},
  mountedPipette[pipetteParts, tipParts]
 ]
```

Volume

We don't do volume because for a pipette tip, we're working with the *outside* dimensions, not the inside

Height and Depth

```
outsideRadiusFromDepth[p : pipette[parts__], depth_] := outsideRadiusFromDepth[bottomToTop @@ Reverse[{parts}], depth]
outsideRadiusFromDepth[tip : pipetteTip[parts__], depth_] := outsideRadiusFromDepth[bottomToTop @@ Reverse[{parts}], depth]
outsideRadiusFromDepth[tip : mountedPipette[parts__], depth_] := outsideRadiusFromDepth[bottomToTop @@ Reverse[{parts}], depth]

outsideRadiusFromDepth[p : bottomToTop[parts__], depth_] := Module[{partCount, heights, cumHeights, radii},
  partCount = Length[{parts}];
  heights = height /@ {parts};
  cumHeights = FoldList[Plus, 0, heights][[1 ;; partCount]];
  Piecewise @ ({{Indeterminate, depth < 0}} ~Join~ MapThread[
      Function[{i, part, height, cumHeight}, {radiusFromDepth[part, depth - cumHeight], Or[i == partCount, depth ≤ cumHeight + height] }],
      {Range[partCount], {parts}, heights, cumHeights} ])
 ]

test @ outsideRadiusFromDepth[pipetteTip[invertedFrustum[h2, rbig, rsmall], invertedCone[h1, rsmall]], depth];
```

$$\text{outsideRadiusFromDepth[pipetteTip[invertedFrustum[h2, rbig, rsmall], invertedCone[h1, rsmall]], depth]} \rightarrow \begin{cases} \text{Indeterminate} & \text{depth} < 0 \\ \frac{\text{depth rsmall}}{\text{h1}} & \text{depth} \leq \text{h1} \\ \text{rbig} + \frac{(\text{depth}-\text{h1}-\text{h2})\,(\text{rbig}-\text{rsmall})}{\text{h2}} & \text{True} \end{cases}$$

Testing

```
Clear[plotProfile]
plotProfile[tipOrPipette : pipetteTip[___] | pipette[___] | mountedPipette[___]] :=
 Plot[outsideRadiusFromDepth[tipOrPipette, depth], {depth, 0, height[tipOrPipette]},
  AspectRatio → outsideRadiusFromDepth[tipOrPipette, height[tipOrPipette]] / height[tipOrPipette] ,
  ImageSize → Full, AxesOrigin → {0, 0}
 ]
plotProfile[other_] :=
 Plot[radiusFromDepth[other, depth], {depth, 0, height[other]}, AspectRatio → radiusFromDepth[other, height[other]] / height[other] ,
  ImageSize → Full, AxesOrigin → {0, 0}
 ]
```

Modelling Specific Labware Types

With specific shapes in hand, we now proceed to model various specific kinds of labware.

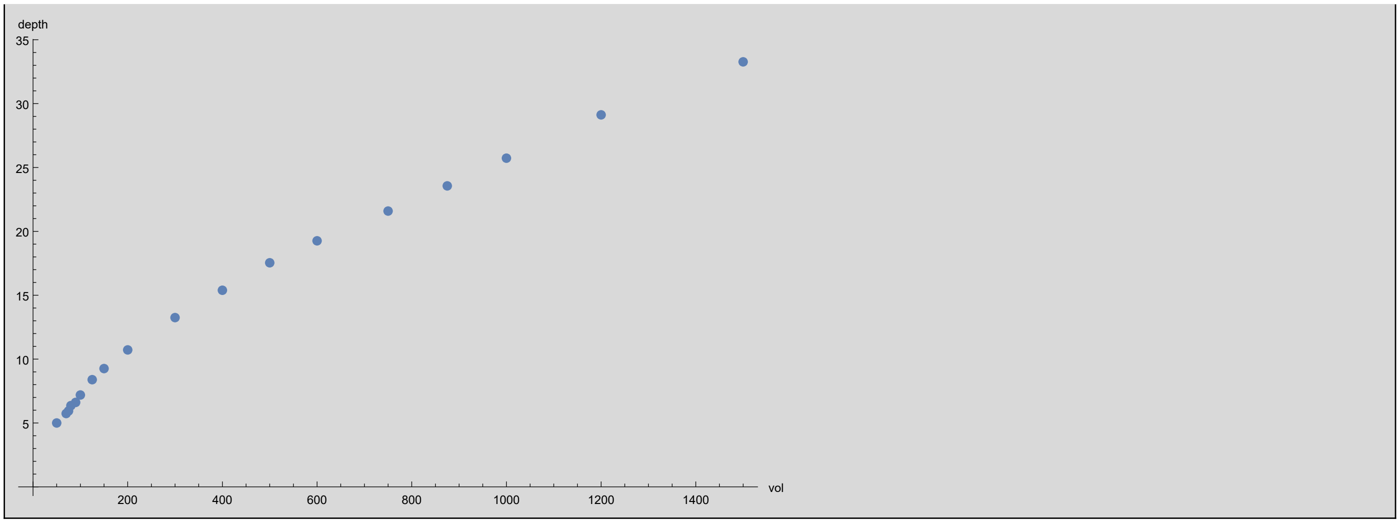
Eppendorf Tubes, 1.5ml and 5.0ml

Data

Data for each of the volume-to-liquid-depth measurements was obtained by pipetting a known volume of food-coloring-colored water, then measuring the depth with a micrometer.

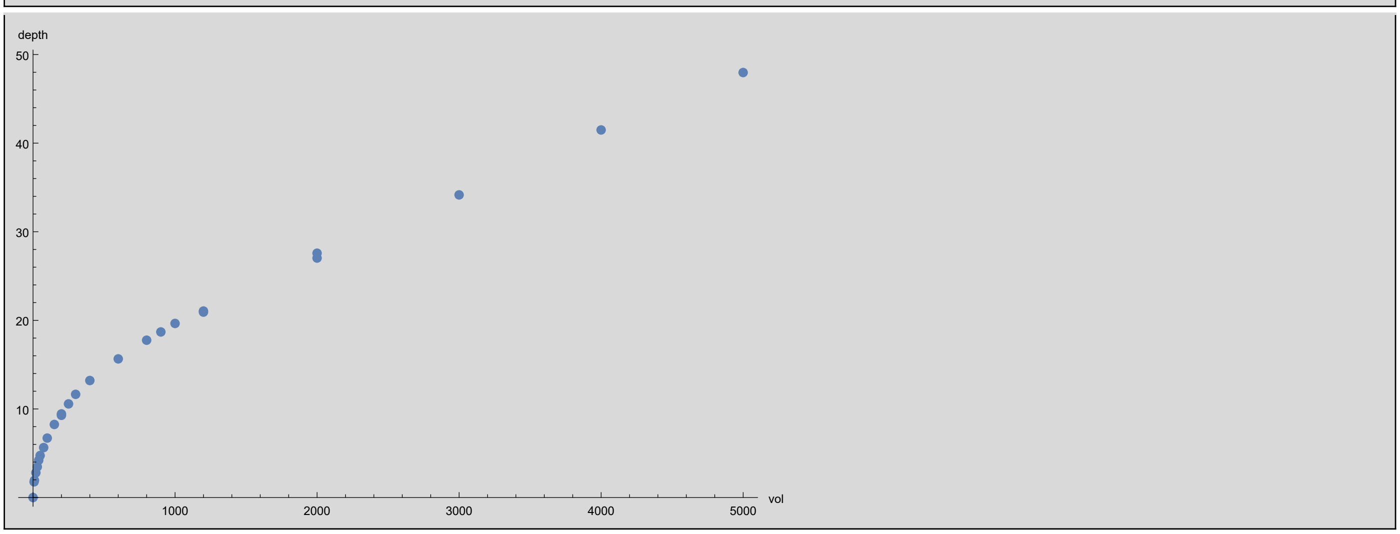
```
eppendorf15Data = ArrayReshape[{50, 5, 70, 5.74, 75, 5.94, 80, 6.36, 90, 6.61, 100, 7.19, 125, 8.39, 150, 9.26, 200, 10.72,
    300, 13.25, 400, 15.39, 500, 17.54, 600, 19.26, 750, 21.59, 875, 23.56, 1000, 25.73, 1200, 29.12, 1500, 33.27}, {18, 2}]
ListPlot[eppendorf15Data, ImageSize → Large, AxesLabel → {"vol", "depth"}, PlotRange → All, AxesOrigin → {0, 0}]
```

```
{{50, 5}, {70, 5.74}, {75, 5.94}, {80, 6.36}, {90, 6.61}, {100, 7.19}, {125, 8.39}, {150, 9.26}, {200, 10.72}, {300, 13.25},
 {400, 15.39}, {500, 17.54}, {600, 19.26}, {750, 21.59}, {875, 23.56}, {1000, 25.73}, {1200, 29.12}, {1500, 33.27}}
```




```
eppendorf50Data = {{200, 9.28`}, {400, 13.21`}, {800, 17.76`}, {1200, 21.05`}, {1000, 19.65`}, {2000, 27.58`}, {3000, 34.16`}, {50, 4.74`},
   {100, 6.71`}, {150, 8.25`}, {200, 9.44`}, {250, 10.57`}, {300, 11.65`}, {600, 15.65`}, {900, 18.69`}, {1200, 20.93`}, {75, 5.64`},
   {40, 4.21`}, {30, 3.47`}, {20, 2.8`}, {10, 1.94`}, {7.5`, 1.77`}, {5000, 47.97}, {4000, 41.49}, {2000, 27.03}, {0, 0}}
ListPlot[eppendorf50Data, ImageSize → Large, AxesLabel → {"vol", "depth"}, PlotRange → All]
```

```
{{200, 9.28}, {400, 13.21}, {800, 17.76}, {1200, 21.05}, {1000, 19.65}, {2000, 27.58}, {3000, 34.16}, {50, 4.74},
 {100, 6.71}, {150, 8.25}, {200, 9.44}, {250, 10.57}, {300, 11.65}, {600, 15.65}, {900, 18.69}, {1200, 20.93}, {75, 5.64},
 {40, 4.21}, {30, 3.47}, {20, 2.8}, {10, 1.94}, {7.5, 1.77}, {5000, 47.97}, {4000, 41.49}, {2000, 27.03}, {0, 0}}
```



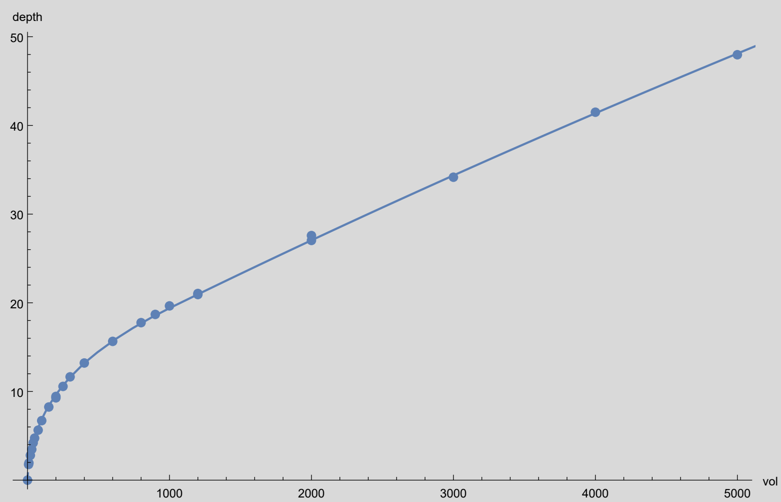
Fitting

```
Clear[fitEppendorfData]

fitEppendorfData[eppendorfData_, specRules_, conicalThreshold_, cylindricalThreshold_, cylConstraints_, coneConstraints_, tubeConstraints_,
  tubeCap_, maxIterations_ : 100] := Block[{hTot, rmid, rBottom, wallBottom, hCyl, hCone, hCap, αCylinder, αCone},
  Module[
   {depthFunc, fit, showFit, zeroify, conicalData, conePart, coneRules,
    angledCone, cylinderData, offsetConicalData, offsetCylinderData, cylinderPart, cylinderRules, rtop, rbottom,
    angledCylinder, tube, α, tubeRules, rconeBig, rconeSmall, rules, rCap, fittedTube, tubeCylinder, tubeCone},
   depthFunc[part_] := Module[{expr, v},
     expr = depthFromVolume[part, v];
     depthFunc[part] = Function[{vol}, expr /. {v → vol}]];
   fit[part_, assump_, vars_, data_] := Module[{errors, err, min, fitRules, asses},
     errors = Function[{vol, depth},
          (depthFunc[part][vol] - depth) ^2
        ] @@ # & /@ data;
     err = Total[errors] // N;
     asses = assumptions[part] && (And @@ assump);
     {min, fitRules} = NMinimize[{err, asses }, vars, MaxIterations → maxIterations];
     fitRules];
   showFit[part_, data_] := Module[{v},
     Show[ListPlot[{data}, ImageSize → Large, AxesLabel → {"vol", "depth"}, PlotRange → All, AxesOrigin → {0, 0}],
      Plot[depthFromVolume[part, v], {v, 0, volume[part]}]]];
   zeroify[data_] := Module[{xMin, yMin},
     {xMin, yMin} = Map[Min, Transpose @ data, {1}];
     Transpose[Transpose[data] - {xMin, yMin}]];

   conicalData = Select[eppendorfData, #[[1]] ≤ conicalThreshold &];
   cylinderData = Select[eppendorfData, #[[1]] >= cylindricalThreshold &];
   offsetConicalData = zeroify[conicalData];
   offsetCylinderData = zeroify[cylinderData];
   printCell[specificationSays[specRules]];

   (* fit the cylinder. this gives us the apex angle of the cylinder. we don't yet know its actual height *)
   (* we dont' know rmid because the bottom of cylinderData might not be right at the mid location *)
   cylinderPart = invertedFrustum[hCyl, rtop, rmid](* /. coneRules*);
   cylinderRules = fit[cylinderPart, cylConstraints, {hCyl, rtop, rmid}, offsetCylinderData];
   angledCylinder = toApexAngled[cylinderPart /. cylinderRules];

   (* fit the cone. this gives us the apex angle of the cone *)
   conePart = invertedFrustum[hCone, rconeBig, rconeSmall];
   coneRules = fit[conePart, coneConstraints, {hCone, rconeBig, rconeSmall}, offsetConicalData];
   angledCone = toApexAngled[conePart /. coneRules];

   (* summarize what we know *)
   rules = {αCylinder → apexangle[angledCylinder], αCone → apexangle[angledCone]};

   (* put these together. *)
   tubeCylinder = invertedFrustum[hCyl, rbig[hCyl, rmid, αCylinder, "apexangle"], αCylinder, "apexangle"] /. rules;
   tubeCone = invertedFrustum[hCone, rmid, αCone, "apexangle"] /. rules;

   rules = rules ~ Join ~ { rBottom → rsmall[tubeCone] };

   tube = conicalTestTube[
     (tubeCylinder),
     (tubeCone),
     (tubeCap /. rules)
    ];
   tube = tube /. { hCone → (hTot /. specRules) - hCyl - hCap};
   tubeRules = fit[tube, tubeConstraints, variables[tube], eppendorfData];
   fittedTube = toCartesian[tube /. tubeRules];
   printCell @ showFit[fittedTube, eppendorfData];
   fittedTube
  ]]
```

We fit the 5.0mL tube

```
fitEppendorf50Data[data_] := Block[{hTot, rmid, wallBottom, rBottom, hCyl, hCone, hCap, volCap, rCap},
  fitEppendorfData[data,
   { hTot → 55.4, rmid → 13.3 / 2, wallBottom → 56.7 - 55.4 },
   1000, 1500,
   { hCyl > 30 }, { hCone > 13 }, {hCap < 2, hCyl > 30, rmid > 6.2, rmid < 6.9},
   invertedSphericalCap[hCap, rBottom, "rCap"]
  ]
 ]
fittedEppendorf5$0M0 = fitEppendorf50Data[eppendorf50Data]
test @ height @ fittedEppendorf5$0M0;
test @ depthFromVolume[fittedEppendorf5$0M0, volume[fittedEppendorf5$0M0]];
test @ volume @ fittedEppendorf5$0M0;
```

```
specificationSays[{hTot → 55.4, rmid → 6.65, wallBottom → 1.3}]
```




```
conicalTestTube[invertedFrustum[35.8967, 7.08628, 6.37479],
 invertedFrustum[18.3424, 6.37479, 1.50899], invertedSphericalCap[1.16088, 1.50899, rCap]]
```

```
height[fittedEppendorf5$0M0] → 55.4
```

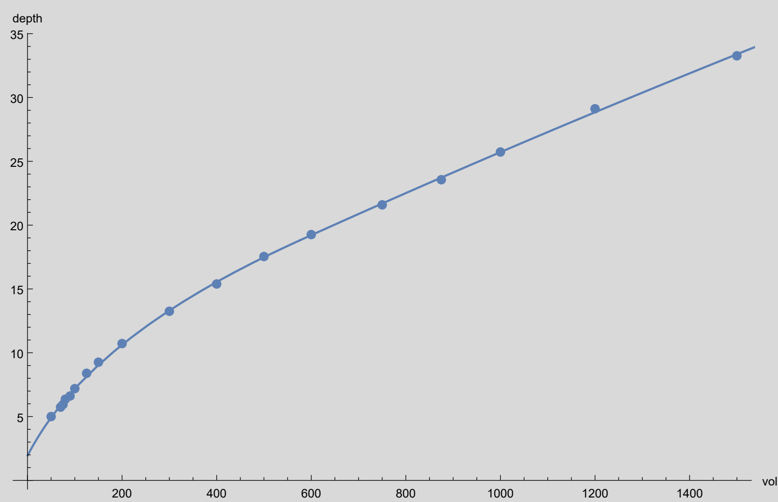
```
depthFromVolume[fittedEppendorf5$0M0, volume[fittedEppendorf5$0M0]] → 55.4
```

```
volume[fittedEppendorf5$0M0] → 6127.44
```

The M0 for the 1.5mL tube is the fitting we used in initial versions; we’ve since found better.

```
fitEppendorf15DataM0[data_] := Block[{hTot, rmid, wallBottom, rBottom, hCyl, hCone, hCap, volCap},
  fitEppendorfData[data, { hTot → 37.8, rmid → 8.7 / 2, wallBottom → 38.9 - 37.8}, 500, 500,
   {hCyl > 12}, {hCone > 10}, {hCap < 5, hCyl > 10, rmid > 4, rmid < 6(*, rCap ≥ hCap*)},
   unknownShape[hCap, volCap]
  ]
 ]
fittedEppendorf1$5M0 = fitEppendorf15DataM0[eppendorf15Data]
test @ height @ fittedEppendorf1$5M0;
test @ depthFromVolume[fittedEppendorf1$5M0, volume[fittedEppendorf1$5M0]];
test @ volume @ fittedEppendorf1$5M0;
```

```
specificationSays[{hTot → 37.8, rmid → 4.35, wallBottom → 1.1}]
```




```
conicalTestTube[invertedFrustum[18.9894, 4.70751, 4.35636], invertedFrustum[16.8419, 4.35636, 2.1099], unknownShape[1.96866, 0.550217]]
```

```
height[fittedEppendorf1$5M0] → 37.8
```

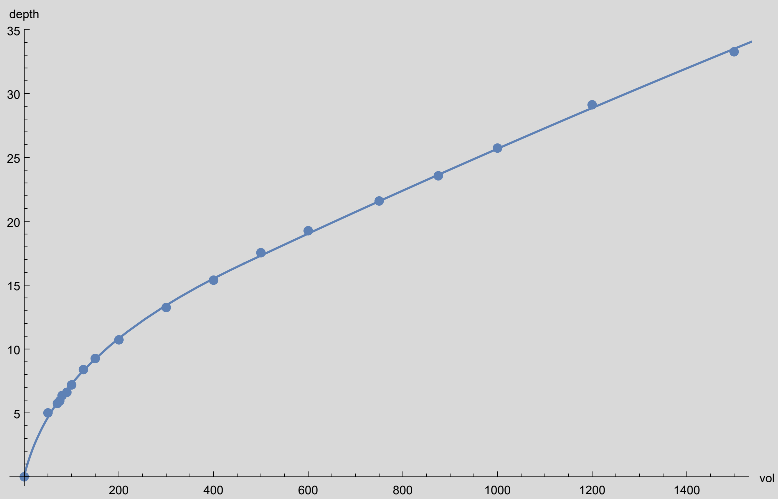
```
depthFromVolume[fittedEppendorf1$5M0, volume[fittedEppendorf1$5M0]] → 37.8
```

```
volume[fittedEppendorf1$5M0] → 1801.76
```

For a revised fitting, we both include the point (0,0) and fit a cap instead of something unknown.

```
fitEppendorf15DataM1[data_] := Block[{hTot, rmid, wallBottom, rBottom, hCyl, hCone, hCap, volCap},
  fitEppendorfData[data, { hTot → 37.8, rmid → 8.7 / 2, wallBottom → 38.9 - 37.8}, 500, 500,
   {hCyl > 12}, {hCone > 10}, {hCap < 5, hCyl > 10, rmid > 4, rmid < 6(*, rCap ≥ hCap*)},
   invertedSphericalCap[hCap, rBottom, "rCap"],
   300
  ]
 ]
fittedEppendorf1$5M1 = fitEppendorf15DataM1[eppendorf15Data ~Join~ {{0, 0}}]
test @ height @ fittedEppendorf1$5M1;
test @ depthFromVolume[fittedEppendorf1$5M1, volume[fittedEppendorf1$5M1]];
test @ volume @ fittedEppendorf1$5M1;
```

```
specificationSays[{hTot → 37.8, rmid → 4.35, wallBottom → 1.1}]
```




```
conicalTestTube[invertedFrustum[21.1258, 4.66267, 4.272],
 invertedFrustum[16.4801, 4.272, 1.48612], invertedSphericalCap[0.194089, 1.48612, rCap]]
```

```
height[fittedEppendorf1$5M1] → 37.8
```

```
depthFromVolume[fittedEppendorf1$5M1, volume[fittedEppendorf1$5M1]] → 37.8
```

```
volume[fittedEppendorf1$5M1] → 1788.68
```

It should be noted that the specification indicates that the upper ‘cylindrical’ inverted frustum isn’t actually an inverted frustum but has a bit of a flare at the top. We ignore that.

Bio-rad Deep Well Plates

The well-plates have proven the most difficult to model, as they are both small compared to other wells and they are enclosed in a grid and a skirt. We have several attempts here, exhibiting the history of the modelling, but the one actually used is the last, which fitted depth-data gathered from absorbance measurements of various volumes of Allura Red.

V1

The Bio-rad specs aren’t internally consistent: there’s a conflict between the well diameters and height vs the well angle. Update: it’s now known that apparent discrepancy arises from the fact that the wells in fact have a capacity larger than 200 μL.

We first choose to honor the well bottom width (2.64).

```
modelBioRad1[] := Module[{cylinderPart, cylinderRules, capacity, hCyl, rtop, rmid, rbottom, conePart,
    hCone, specRules, rules, hTot, wallBottom, αCone, tube, vol, solns, soln, assumpts, constraint, extra, hCylMin },

   (* we assume the top is an actual cylinder rather than an inverted frustum *)
   cylinderPart = cylinder[hCyl, rtop];
   cylinderRules = {rmid → rtop};

   conePart = invertedFrustum[hCone, rmid, αCone, "apexangle"]; (* doesn't honor rbottom on its own *)
   conePart = invertedFrustum[hCone, rmid, rbottom];

   specRules = { hTot → 14.81, rtop → 5.46 / 2, rbottom → 2.64 / 2, wallBottom → 16.06 - 14.81, αCone → toRadian[17.5] / 2};
   (*printCell[specificationSays[specRules]];*)
   tube = conicalTestTube[cylinderPart, conePart, emptyCylinder[]];
   rules = {hCone → hTot - hCyl } ~Join~ cylinderRules ~Join~ specRules;
   tube = tube //. rules;
   vol = volume[tube];
   capacity = 200 ;
   assumpts = True;
   solns = Solve[vol == capacity && assumpts, {hCyl}];
   soln = First @ solns;
   tube //. soln // toCartesian
  ]
modelledBioRad1 = modelBioRad1[];
test @ modelledBioRad1;
test @ toDeg[apexangle[parts[modelledBioRad1]["conical"]] * 2];
test @ (2 * rsmall[parts[modelledBioRad1]["conical"]]);
```

```
modelledBioRad1 → conicalTestTube[cylinder[0.150026, 2.73], invertedFrustum[14.66, 2.73, 1.32], cylinder[0, 0]]
```

```
toDeg[apexangle[parts[modelledBioRad1][conical]] 2] → 10.9876
```

```
2 rsmall[parts[modelledBioRad1][conical]] → 2.64
```

V2

So instead we honor the apex angle of the cone (17.5°).

```
modelBioRad2[] := Module[{cylinderPart, cylinderRules, capacity, hCyl, rtop, rmid, rbottom, conePart,
    hCone, specRules, rules, hTot, wallBottom, αCone, tube, vol, solns, soln, assumpts, constraint, extra, hCylMin },

   (* we assume the top is an actual cylinder rather than an inverted frustum *)
   cylinderPart = cylinder[hCyl, rtop];
   cylinderRules = {rmid → rtop};

   conePart = invertedFrustum[hCone, rmid, αCone, "apexangle"]; (* doesn't honor rbottom on its own *)

   specRules = { hTot → 14.81, rtop → 5.46 / 2, rbottom → 2.64 / 2, wallBottom → 16.06 - 14.81, αCone → toRadian[17.5] / 2};
   (*printCell[specificationSays[specRules]];*)
   tube = conicalTestTube[cylinderPart, conePart, emptyCylinder[]];
   rules = {hCone → hTot - hCyl } ~Join~ cylinderRules ~Join~ specRules;
   tube = tube //. rules;
   vol = volume[tube];
   capacity = 200 ;
   assumpts = hCyl > 0 && hCyl < 5;
   solns = Solve[vol == capacity && assumpts, {hCyl}];
   soln = First @ solns;
   tube //. soln // toCartesian
  ]
modelledBioRad2 = modelBioRad2[];
test @ modelledBioRad2;
test @ toDeg[apexangle[parts[modelledBioRad2]["conical"]] * 2];
test @ (2 * rsmall[parts[modelledBioRad2]["conical"]]);
```

```
modelledBioRad2 → conicalTestTube[cylinder[2.83192, 2.73], invertedFrustum[11.9781, 2.73, 0.886397], cylinder[0, 0]]
```

```
toDeg[apexangle[parts[modelledBioRad2][conical]] 2] → 17.5
```

```
2 rsmall[parts[modelledBioRad2][conical]] → 1.77279
```

V3

Next, we honor *both* the apex angle and the bottom dimension. But to do that, we need to admit that the capacity of the well is greater than stated (which is almost certainly true).

```
modelBioRad3[] := Module[{cylinderPart, cylinderRules, capacity, hCyl, rtop, rmid, rbottom, conePart, hCone,
    specRules, rules, hTot, wallBottom, αCone, tube, vol, solns, soln, assumpts, constraint, extra, hCylMin, hCylSoln },

   (* we assume the top is an actual cylinder rather than an inverted frustum *)
   cylinderPart = cylinder[hCyl, rtop];
   cylinderRules = {rmid → rtop};

   conePart = invertedFrustum[hCone, rmid, αCone, "apexangle"]; (* doesn't honor rbottom on its own *)

   specRules = { hTot → 14.81, rtop → 5.46 / 2, rbottom → 2.64 / 2, wallBottom → 16.06 - 14.81, αCone → toRadian[17.5] / 2};
   (*printCell[specificationSays[specRules]];*)
   tube = conicalTestTube[cylinderPart, conePart, emptyCylinder[]];
   rules = {hCone → hTot - hCyl } ~ Join ~ cylinderRules ~ Join ~ specRules;
   tube = tube //. rules;

   constraint = (rsmall[conePart] - rbottom) //. rules;
   hCylSoln = First @ Solve[constraint == 0, {hCyl}];
   tube = tube //. hCylSoln;

   vol = volume[tube];
   capacity = 200 + extra;
   assumpts = extra ≥ 0;
   solns = Solve[vol == capacity && assumpts, {extra}];
   soln = First @ solns;
   tube //. soln // toCartesian
  ]
modelledBioRad3 = modelBioRad3[];
test @ modelledBioRad3;
test @ toDeg[apexangle[parts[modelledBioRad3]["conical"]] * 2];
test @ (2 * rsmall[parts[modelledBioRad3]["conical"]]);
test @ (2 * rbig[parts[modelledBioRad3]["conical"]]);
test @ volume[modelledBioRad3];
```

```
modelledBioRad3 → conicalTestTube[cylinder[5.64908, 2.73], invertedFrustum[9.16092, 2.73, 1.32], cylinder[0, 0]]
```

```
toDeg[apexangle[parts[modelledBioRad3][conical]] 2] → 17.5
```

```
2 rsmall[parts[modelledBioRad3][conical]] → 2.64
```

```
2 rbig[parts[modelledBioRad3][conical]] → 5.46
```

```
volume[modelledBioRad3] → 255.051
```

V4

In our fourth attempt, we use the experimentally-measured capacity volume of the well.

```
modelBioRad4[] := Module[{cylinderPart, cylinderRules, capacity, hCyl, rtop, rmid, rbottom, conePart, hCone, specRules,
   rules, hTot, wallBottom, αCone, tube, volConstraint, solns, soln, assumpts, rConstraint, extra, hCylMin, hCylSoln },

  cylinderPart = invertedFrustum[hCyl, rtop, rmid];
  cylinderRules = {};

  conePart = invertedFrustum[hCone, rmid, αCone, "apexangle"]; (* doesn't honor rbottom on its own *)

  (* note we tweak rtop as well to try account for the flare at the top *)
  specRules = { hTot → 14.81, rtop → 5.4 / 2, rbottom → 2.64 / 2, wallBottom → 16.06 - 14.81, αCone → toRadian[17.5] / 2, capacity → 235};
  (*printCell[specificationSays[specRules]];*)
  tube = conicalTestTube[cylinderPart, conePart, emptyCylinder[]];
  rules = {hCone → hTot - hCyl } ~Join~ cylinderRules ~Join~ specRules;
  tube = tube //. rules;

  rConstraint = (rsmall[conePart] == rbottom) //. rules;
  test @ rConstraint;
  volConstraint = volume[tube] == capacity //. rules;
  test @ volConstraint;
  assumpts = ( hCyl > 0 && rmid > 0 && rmid < rtop (*&& hCyl > 5.7*) (* choose the non-cylinder cylinderPart *) ) //. rules;
  solns = Solve[rConstraint && volConstraint && assumpts, {rmid, hCyl}];
  test @ solns;
  soln = First @ solns;
  tube //. soln // toCartesian
 ]
modelledBioRad4 = modelBioRad4[];
test @ modelledBioRad4;
test @ toDeg[apexangle[parts[modelledBioRad4]["conical"]] * 2];
test @ (2 * rsmall[parts[modelledBioRad4]["conical"]]);
test @ (2 * rbig[parts[modelledBioRad4]["conical"]]);
test @ (2 * rsmall[parts[modelledBioRad4]["cylindrical"]]);
test @ (2 * rbig[parts[modelledBioRad4]["cylindrical"]]);
test @ volume[modelledBioRad4];
```

```
rConstraint$67840 → -0.153915 (14.81 - hCyl$67840) + rmid$67840 == 1.32
```

```
volConstraint$67840 →
 6.80375 rmid$67840^3 - 0.0248078 (-14.81 + hCyl$67840 + 6.4971 rmid$67840)^3 + hCyl$67840 (7.63407 + rmid$67840 (2.82743 + 1.0472 rmid$67840)) == 235
```

... Solve: Solve was unable to solve the system with inexact coefficients. The answer was obtained by solving a corresponding exact system and numericizing the result.

```
solns$67840 → {{rmid$67840 → 2.33872, hCyl$67840 → 8.1913}}
```

```
modelledBioRad4 → conicalTestTube[invertedFrustum[8.1913, 2.7, 2.33872], invertedFrustum[6.6187, 2.33872, 1.32], cylinder[0, 0]]
```

```
toDeg[apexangle[parts[modelledBioRad4][conical]] 2] → 17.5
```

```
2 rsmall[parts[modelledBioRad4][conical]] → 2.64
```

```
2 rbig[parts[modelledBioRad4][conical]] → 4.67743
```

```
2 rsmall[parts[modelledBioRad4][cylindrical]] → 4.67743
```

```
2 rbig[parts[modelledBioRad4][cylindrical]] → 5.4
```

```
volume[modelledBioRad4] → 235.
```

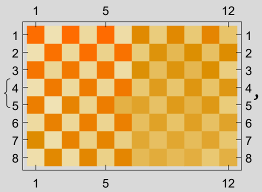
The height of the cylindrical part here (hCyl) seems unreasonably large, given the observed dimensions of the tubes. For the moment, at least, we don't use this approach.

V5

This analyzes the results of experiment E19110201. We made a plate with a patchwork of various volumes of Allura Red in water and adjacent water controls. We read the (one) plate six times on the plate reader at 504nm, three times at 0° and three times at 180°, in an attempt to even out the variation in plate reader readings across the plate.

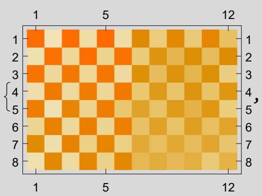
Load the plates and canonicalize orientation of plate.

```
plate1 = {{1.388`, 0.398`, 1.407`, 0.43`, 1.414`, 0.425`, 0.972`, 0.516`, 0.981`, 0.535`, 1.031`, 0.546`},
   {0.399`, 1.325`, 0.436`, 1.347`, 0.464`, 1.311`, 0.522`, 0.859`, 0.582`, 0.897`, 0.553`, 0.936`},
   {1.234`, 0.443`, 1.178`, 0.437`, 1.194`, 0.427`, 0.892`, 0.564`, 0.848`, 0.591`, 0.897`, 0.507`},
   {0.408`, 1.159`, 0.431`, 1.129`, 0.441`, 1.174`, 0.539`, 0.81`, 0.543`, 0.844`, 0.547`, 0.843`},
   {1.115`, 0.397`, 1.036`, 0.461`, 1.122`, 0.636`, 0.748`, 0.552`, 0.818`, 0.627`, 0.832`, 0.518`},
   {0.429`, 1.081`, 0.437`, 1.111`, 0.452`, 1.102`, 0.559`, 0.756`, 0.547`, 0.769`, 0.553`, 0.748`},
   {0.947`, 0.416`, 1.046`, 0.48`, 1.032`, 0.501`, 0.723`, 0.554`, 0.726`, 0.565`, 0.734`, 0.541`},
   {0.412`, 0.989`, 0.428`, 1.009`, 0.479`, 1.012`, 0.576`, 0.677`, 0.575`, 0.649`, 0.536`, 0.637`}};
{MatrixPlot[plate1], TableForm[plate1]}
```



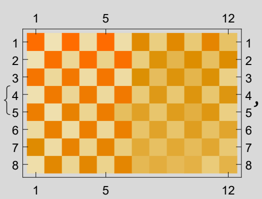
1.388	0.398	1.407	0.43	1.414	0.425	0.972	0.516	0.981	0.535	1.031	0.546
0.399	1.325	0.436	1.347	0.464	1.311	0.522	0.859	0.582	0.897	0.553	0.936
1.234	0.443	1.178	0.437	1.194	0.427	0.892	0.564	0.848	0.591	0.897	0.507
0.408	1.159	0.431	1.129	0.441	1.174	0.539	0.81	0.543	0.844	0.547	0.843
1.115	0.397	1.036	0.461	1.122	0.636	0.748	0.552	0.818	0.627	0.832	0.518
0.429	1.081	0.437	1.111	0.452	1.102	0.559	0.756	0.547	0.769	0.553	0.748
0.947	0.416	1.046	0.48	1.032	0.501	0.723	0.554	0.726	0.565	0.734	0.541
0.412	0.989	0.428	1.009	0.479	1.012	0.576	0.677	0.575	0.649	0.536	0.637

```
plate2 = {{1.391`, 0.397`, 1.41`, 0.428`, 1.413`, 0.423`, 0.974`, 0.519`, 0.986`, 0.53`, 1.025`, 0.548`},
   {0.403`, 1.325`, 0.441`, 1.341`, 0.46`, 1.303`, 0.528`, 0.854`, 0.592`, 0.899`, 0.563`, 0.926`},
   {1.237`, 0.441`, 1.177`, 0.443`, 1.197`, 0.433`, 0.891`, 0.556`, 0.858`, 0.595`, 0.892`, 0.506`},
   {0.412`, 1.163`, 0.434`, 1.127`, 0.444`, 1.171`, 0.534`, 0.824`, 0.54`, 0.854`, 0.543`, 0.847`},
   {1.115`, 0.398`, 1.037`, 0.461`, 1.122`, 0.535`, 0.748`, 0.544`, 0.818`, 0.635`, 0.845`, 0.515`},
   {0.447`, 1.074`, 0.439`, 1.108`, 0.453`, 1.097`, 0.547`, 0.761`, 0.544`, 0.783`, 0.55`, 0.746`},
   {0.95`, 0.414`, 1.056`, 0.484`, 1.023`, 0.495`, 0.713`, 0.55`, 0.719`, 0.562`, 0.732`, 0.537`},
   {0.407`, 0.997`, 0.431`, 1.011`, 0.477`, 1.005`, 0.576`, 0.671`, 0.574`, 0.649`, 0.531`, 0.636`}};
{MatrixPlot[plate2], TableForm[plate2]}
```



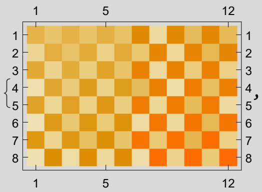
1.391	0.397	1.41	0.428	1.413	0.423	0.974	0.519	0.986	0.53	1.025	0.548
0.403	1.325	0.441	1.341	0.46	1.303	0.528	0.854	0.592	0.899	0.563	0.926
1.237	0.441	1.177	0.443	1.197	0.433	0.891	0.556	0.858	0.595	0.892	0.506
0.412	1.163	0.434	1.127	0.444	1.171	0.534	0.824	0.54	0.854	0.543	0.847
1.115	0.398	1.037	0.461	1.122	0.535	0.748	0.544	0.818	0.635	0.845	0.515
0.447	1.074	0.439	1.108	0.453	1.097	0.547	0.761	0.544	0.783	0.55	0.746
0.95	0.414	1.056	0.484	1.023	0.495	0.713	0.55	0.719	0.562	0.732	0.537
0.407	0.997	0.431	1.011	0.477	1.005	0.576	0.671	0.574	0.649	0.531	0.636

```
plate3 = {{1.391`, 0.397`, 1.407`, 0.427`, 1.412`, 0.422`, 0.97`, 0.517`, 0.979`, 0.534`, 1.026`, 0.547`},
   {0.4`, 1.323`, 0.455`, 1.342`, 0.482`, 1.329`, 0.525`, 0.881`, 0.62`, 0.921`, 0.552`, 0.928`},
   {1.236`, 0.449`, 1.197`, 0.433`, 1.227`, 0.434`, 0.893`, 0.556`, 0.878`, 0.62`, 0.891`, 0.502`},
   {0.411`, 1.162`, 0.457`, 1.156`, 0.445`, 1.173`, 0.533`, 0.851`, 0.537`, 0.876`, 0.542`, 0.871`},
   {1.115`, 0.413`, 1.06`, 0.463`, 1.121`, 0.553`, 0.77`, 0.543`, 0.817`, 0.66`, 0.869`, 0.512`},
   {0.439`, 1.08`, 0.434`, 1.109`, 0.453`, 1.099`, 0.546`, 0.76`, 0.544`, 0.806`, 0.549`, 0.747`},
   {0.963`, 0.433`, 1.057`, 0.473`, 1.042`, 0.526`, 0.712`, 0.55`, 0.718`, 0.561`, 0.732`, 0.536`},
   {0.421`, 0.998`, 0.458`, 1.004`, 0.473`, 1.007`, 0.578`, 0.674`, 0.574`, 0.648`, 0.531`, 0.635`}};
{MatrixPlot[plate3], TableForm[plate3]}
```



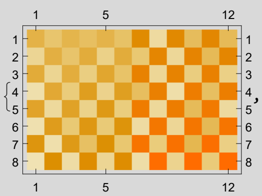
1.391	0.397	1.407	0.427	1.412	0.422	0.97	0.517	0.979	0.534	1.026	0.547
0.4	1.323	0.455	1.342	0.482	1.329	0.525	0.881	0.62	0.921	0.552	0.928
1.236	0.449	1.197	0.433	1.227	0.434	0.893	0.556	0.878	0.62	0.891	0.502
0.411	1.162	0.457	1.156	0.445	1.173	0.533	0.851	0.537	0.876	0.542	0.871
1.115	0.413	1.06	0.463	1.121	0.553	0.77	0.543	0.817	0.66	0.869	0.512
0.439	1.08	0.434	1.109	0.453	1.099	0.546	0.76	0.544	0.806	0.549	0.747
0.963	0.433	1.057	0.473	1.042	0.526	0.712	0.55	0.718	0.561	0.732	0.536
0.421	0.998	0.458	1.004	0.473	1.007	0.578	0.674	0.574	0.648	0.531	0.635

```
plate4 = {{0.636`, 0.56`, 0.641`, 0.564`, 0.65`, 0.567`, 1.019`, 0.491`, 1.04`, 0.582`, 1.081`, 0.591`},
   {0.503`, 0.676`, 0.565`, 0.699`, 0.524`, 0.696`, 0.549`, 1.093`, 0.496`, 1.113`, 0.593`, 1.083`},
   {0.705`, 0.506`, 0.693`, 0.53`, 0.75`, 0.564`, 1.135`, 0.494`, 1.124`, 0.532`, 1.159`, 0.527`},
   {0.468`, 0.726`, 0.536`, 0.819`, 0.516`, 0.813`, 0.574`, 1.17`, 0.476`, 1.21`, 0.572`, 1.193`},
   {0.77`, 0.499`, 0.785`, 0.522`, 0.848`, 0.531`, 1.22`, 0.497`, 1.209`, 0.585`, 1.256`, 0.49`},
   {0.445`, 0.837`, 0.543`, 0.859`, 0.537`, 0.886`, 0.486`, 1.305`, 0.495`, 1.317`, 0.592`, 1.352`},
   {0.859`, 0.504`, 0.855`, 0.57`, 0.931`, 0.556`, 1.366`, 0.541`, 1.42`, 0.613`, 1.45`, 0.528`},
   {0.447`, 0.937`, 0.48`, 0.98`, 0.495`, 0.978`, 0.478`, 1.5`, 0.511`, 1.516`, 0.538`, 1.523`}};
{MatrixPlot[plate4], TableForm[plate4]}
```



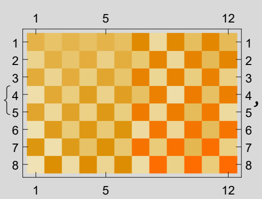
0.636	0.56	0.641	0.564	0.65	0.567	1.019	0.491	1.04	0.582	1.081	0.591
0.503	0.676	0.565	0.699	0.524	0.696	0.549	1.093	0.496	1.113	0.593	1.083
0.705	0.506	0.693	0.53	0.75	0.564	1.135	0.494	1.124	0.532	1.159	0.527
0.468	0.726	0.536	0.819	0.516	0.813	0.574	1.17	0.476	1.21	0.572	1.193
0.77	0.499	0.785	0.522	0.848	0.531	1.22	0.497	1.209	0.585	1.256	0.49
0.445	0.837	0.543	0.859	0.537	0.886	0.486	1.305	0.495	1.317	0.592	1.352
0.859	0.504	0.855	0.57	0.931	0.556	1.366	0.541	1.42	0.613	1.45	0.528
0.447	0.937	0.48	0.98	0.495	0.978	0.478	1.5	0.511	1.516	0.538	1.523

```
plate5 = {{0.645`, 0.559`, 0.633`, 0.573`, 0.641`, 0.568`, 1.019`, 0.49`, 1.037`, 0.588`, 1.079`, 0.595`},
   {0.501`, 0.675`, 0.56`, 0.698`, 0.524`, 0.698`, 0.546`, 1.095`, 0.499`, 1.116`, 0.595`, 1.089`},
   {0.701`, 0.502`, 0.695`, 0.531`, 0.749`, 0.565`, 1.135`, 0.5`, 1.126`, 0.532`, 1.157`, 0.527`},
   {0.469`, 0.728`, 0.539`, 0.819`, 0.521`, 0.818`, 0.574`, 1.167`, 0.477`, 1.213`, 0.573`, 1.194`},
   {0.773`, 0.495`, 0.787`, 0.522`, 0.854`, 0.533`, 1.22`, 0.498`, 1.212`, 0.59`, 1.256`, 0.491`},
   {0.456`, 0.83`, 0.545`, 0.865`, 0.541`, 0.888`, 0.489`, 1.307`, 0.5`, 1.318`, 0.596`, 1.347`},
   {0.879`, 0.469`, 0.853`, 0.572`, 0.933`, 0.548`, 1.368`, 0.539`, 1.422`, 0.616`, 1.449`, 0.528`},
   {0.436`, 0.942`, 0.485`, 0.98`, 0.501`, 0.98`, 0.48`, 1.503`, 0.513`, 1.514`, 0.54`, 1.524`}};
{MatrixPlot[plate5], TableForm[plate5]}
```



0.645	0.559	0.633	0.573	0.641	0.568	1.019	0.49	1.037	0.588	1.079	0.595
0.501	0.675	0.56	0.698	0.524	0.698	0.546	1.095	0.499	1.116	0.595	1.089
0.701	0.502	0.695	0.531	0.749	0.565	1.135	0.5	1.126	0.532	1.157	0.527
0.469	0.728	0.539	0.819	0.521	0.818	0.574	1.167	0.477	1.213	0.573	1.194
0.773	0.495	0.787	0.522	0.854	0.533	1.22	0.498	1.212	0.59	1.256	0.491
0.456	0.83	0.545	0.865	0.541	0.888	0.489	1.307	0.5	1.318	0.596	1.347
0.879	0.469	0.853	0.572	0.933	0.548	1.368	0.539	1.422	0.616	1.449	0.528
0.436	0.942	0.485	0.98	0.501	0.98	0.48	1.503	0.513	1.514	0.54	1.524

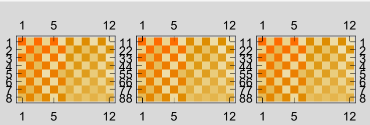
```
plate6 = {{0.648`, 0.57`, 0.636`, 0.578`, 0.64`, 0.585`, 1.018`, 0.491`, 1.037`, 0.593`, 1.078`, 0.601`},
   {0.51`, 0.675`, 0.564`, 0.698`, 0.526`, 0.697`, 0.554`, 1.105`, 0.497`, 1.119`, 0.603`, 1.095`},
   {0.702`, 0.502`, 0.698`, 0.531`, 0.75`, 0.565`, 1.135`, 0.5`, 1.124`, 0.531`, 1.158`, 0.526`},
   {0.468`, 0.73`, 0.539`, 0.821`, 0.521`, 0.819`, 0.578`, 1.168`, 0.475`, 1.219`, 0.578`, 1.193`},
   {0.77`, 0.496`, 0.788`, 0.523`, 0.856`, 0.531`, 1.221`, 0.499`, 1.217`, 0.596`, 1.256`, 0.49`},
   {0.454`, 0.838`, 0.543`, 0.864`, 0.546`, 0.885`, 0.488`, 1.317`, 0.499`, 1.323`, 0.599`, 1.347`},
   {0.88`, 0.466`, 0.851`, 0.571`, 0.933`, 0.556`, 1.371`, 0.541`, 1.422`, 0.618`, 1.45`, 0.528`},
   {0.435`, 0.939`, 0.482`, 0.982`, 0.501`, 0.979`, 0.48`, 1.504`, 0.514`, 1.514`, 0.54`, 1.524`}};
{MatrixPlot[plate6], TableForm[plate6]}
```



0.648	0.57	0.636	0.578	0.64	0.585	1.018	0.491	1.037	0.593	1.078	0.601
0.51	0.675	0.564	0.698	0.526	0.697	0.554	1.105	0.497	1.119	0.603	1.095
0.702	0.502	0.698	0.531	0.75	0.565	1.135	0.5	1.124	0.531	1.158	0.526
0.468	0.73	0.539	0.821	0.521	0.819	0.578	1.168	0.475	1.219	0.578	1.193
0.77	0.496	0.788	0.523	0.856	0.531	1.221	0.499	1.217	0.596	1.256	0.49
0.454	0.838	0.543	0.864	0.546	0.885	0.488	1.317	0.499	1.323	0.599	1.347
0.88	0.466	0.851	0.571	0.933	0.556	1.371	0.541	1.422	0.618	1.45	0.528
0.435	0.939	0.482	0.982	0.501	0.979	0.48	1.504	0.514	1.514	0.54	1.524

We want to normalize all the plates to have the same orientation

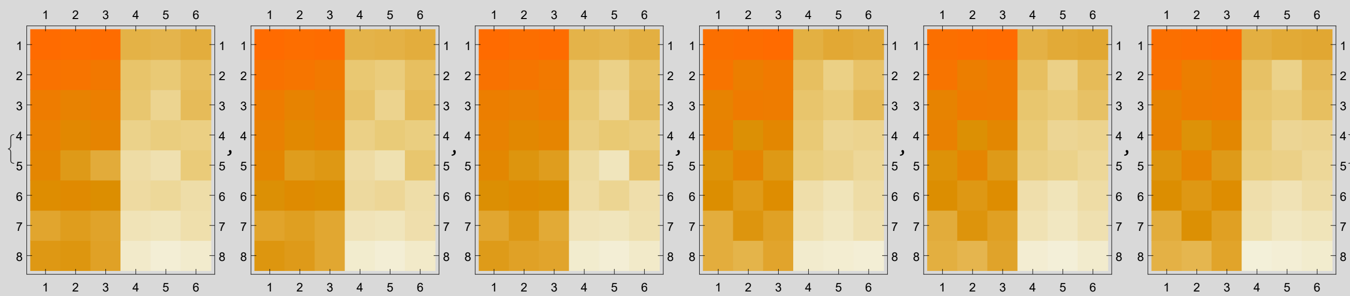
```
Clear[rot90]
rot90[mat_] := Transpose[Reverse[mat, {2}]]
plate4 = rot90 @ rot90 @ plate4;
plate5 = rot90 @ rot90 @ plate5;
plate6 = rot90 @ rot90 @ plate6;
Row @@ {{plate4 // MatrixPlot, plate5 // MatrixPlot, plate6 // MatrixPlot}}
```



We subtract each sample well (containing Allura Red in water) from its immediately horizontally adjacent control well (that contains only water).

```
plates = {plate1, plate2, plate3, plate4, plate5, plate6};
```

```
Clear[baselineSubtractPlate]
baselineSubtractPlate[plate_] := Function[row, Module[{nCols = Length[row]},
    Function[iPair, Abs[row[[2 iPair - 1]] - row[[2 iPair]]]] /@ Range[nCols / 2]
   ]] /@ plate
MatrixPlot @ baselineSubtractPlate[#] & /@ plates
```



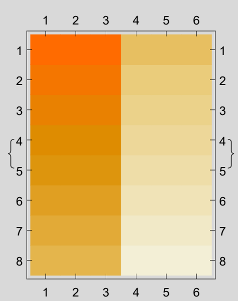
The volumes we pipetted we obtain from the protocol definition.

```
plateVolumes = Reverse @ {5, 10, 15, 20, 25, 30, 35, 50, 60, 70, 80, 90, 100, 125, 150, 175}
plateVolumes = {#, #, #} & /@ plateVolumes
plateVolumes = ArrayReshape[Flatten[Flatten[{plateVolumes[[1 ;; 8, All]], plateVolumes[[9 ;; 16, All]]}, {2}]], {8, 6}]
{MatrixPlot @ plateVolumes}
```

```
{175, 150, 125, 100, 90, 80, 70, 60, 50, 35, 30, 25, 20, 15, 10, 5}
```

```
{{175, 175, 175}, {150, 150, 150}, {125, 125, 125}, {100, 100, 100}, {90, 90, 90}, {80, 80, 80}, {70, 70, 70},
 {60, 60, 60}, {50, 50, 50}, {35, 35, 35}, {30, 30, 30}, {25, 25, 25}, {20, 20, 20}, {15, 15, 15}, {10, 10, 10}, {5, 5, 5}}
```

```
{{175, 175, 175, 50, 50, 50}, {150, 150, 150, 35, 35, 35}, {125, 125, 125, 30, 30, 30},
 {100, 100, 100, 25, 25, 25}, {90, 90, 90, 20, 20, 20}, {80, 80, 80, 15, 15, 15}, {70, 70, 70, 10, 10, 10}, {60, 60, 60, 5, 5, 5}}
```



By Beer's Law, the absorbance of each baseline subtracted plate in each well should be a linear factor times the depth of the well. Specifically, that factor should be the attenuation coefficient of Allura Red times the concentration.

```
attenuation == absorptivity depth concentration
Solve[%, depth]
```

```
attenuation == absorptivity concentration depth
```

$$\left\{\left\{\text{depth} \to \frac{\text{attenuation}}{\text{absorptivity concentration}}\right\}\right\}$$

The concentration of Allura Red is in all wells 32.2 μM.

The absorptivity is understood (see http://www.webpages.uidaho.edu/ifcheng/Chem%20253/labs/Experiment%202%202015-02-13.docx) to be 25,900 M^{-1} cm^{-1}.

```
Quantity[25 900, "per Molar per cm"]
UnitConvert [%, "per microMolar per mm"]
% * Quantity[32.2, "microMolar"]
absorptivityTimesConcentration = QuantityMagnitude[%]
```

```
25 900 / (cm M)
```

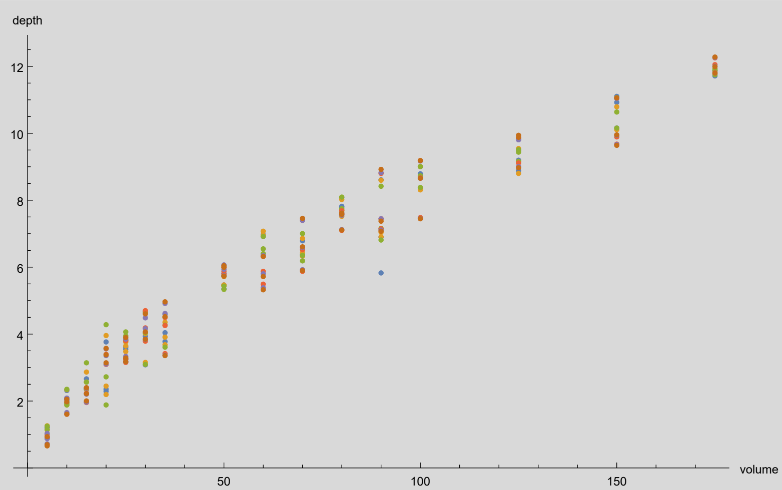
```
  259
─────── / (mm μM)
100 000
```

```
0.083398 /mm
```

```
0.083398
```

Let’s have a quick look at this data.

```
Clear[plateVolumeAndDepth]
plateVolumeAndDepth[plate_] := pairUp[Flatten[plateVolumes], Flatten[baselineSubtractPlate[plate] / absorptivityTimesConcentration]]
plateData = Join @@ (plateVolumeAndDepth /@ plates);
ListPlot[ plateVolumeAndDepth[#] & /@ plates, AxesOrigin → {0, 0}, AxesLabel → {"volume", "depth"}, ImageSize → Large]
(*ListPlot[plateData]*)
```



We model and fit the data

```
Clear[modelBioRad5]
modelBioRad5[] :=
 Block[{cylinderPart, cylinderRules, capacity, hCyl, rtop, rmid, rbottom, conePart, hCone, specRules, rules, hTot, wallBottom, αCone, tube,
   vol, solns, soln, assumpts, constraint, extra, hCylMin, hCylSoln, genericDepth, errors, err, min, tubeRules, data, dataAssumptions },

  data = plateData;
  dataAssumptions = (*20000 < alluraRedAbsorptivity < 30000*) True;

  (* we assume the top is an actual cylinder rather than an inverted frustum *)
  cylinderPart = cylinder[hCyl, rtop];
  cylinderRules = {rmid → rtop};

  conePart = invertedFrustum[hCone, rmid, rbottom];

  (* Here, from the spec we only use the total interior height of the well (which we don't have in our data) *)
  specRules = { hTot → 14.81 };

  (* We model the whole tube, all at once *)
  tube = conicalTestTube[cylinderPart, conePart, emptyCylinder[]];
  rules = {hCone → hTot - hCyl } ~Join~ cylinderRules ~Join~ specRules;
  tube = tube //. rules;

  (* We fit the data *)
  Clear[genericDepth];
  genericDepth[part_] := Module[{expr, v},
    expr = depthFromVolume[part, v];
    genericDepth[part] = Function[{vol}, expr /. {v → vol}]
   ];

  errors = Function[{vol, depth}, (genericDepth[tube][vol] - depth)^2] @@ # & /@ data;
  err = Total[errors] // N;
  {min, tubeRules} = NMinimize[{err, assumptions[tube] && assumptions[tube] && dataAssumptions}, Union[variables[tube], variables[data]]];
  {tubeRules, tube /. tubeRules}
 ]
{modelledBioRad5Rules, modelledBioRad5} = modelBioRad5[];
test @ modelledBioRad5Rules;
test @ modelledBioRad5;
test @ toDeg[apexangle[parts[modelledBioRad5]["conical"]] * 2];
test @ (2 * rsmall[parts[modelledBioRad5]["conical"]]);
test @ (2 * rbig[parts[modelledBioRad5]["conical"]]);
test @ volume[modelledBioRad5];
expr = depthFromVolume[modelledBioRad5, vol]
Show[
 Plot[expr, {vol, 0, 200}, GridLines → Automatic, AxesOrigin → {0, 0}, AxesLabel → {"volume", "depth"}, ImageSize → Large],
 ListPlot[ plateVolumeAndDepth[#] & /@ plates, AxesOrigin → {0, 0}, AxesLabel → {"volume", "depth"}, ImageSize → Large]
]
```

```
modelledBioRad5Rules → {hCyl → 6.69498, rbottom → 1.16608, rtop → 2.61859}
```

```
modelledBioRad5 → conicalTestTube[cylinder[6.69498, 2.61859], invertedFrustum[8.11502, 2.61859, 1.16608], cylinder[0, 0]]
```

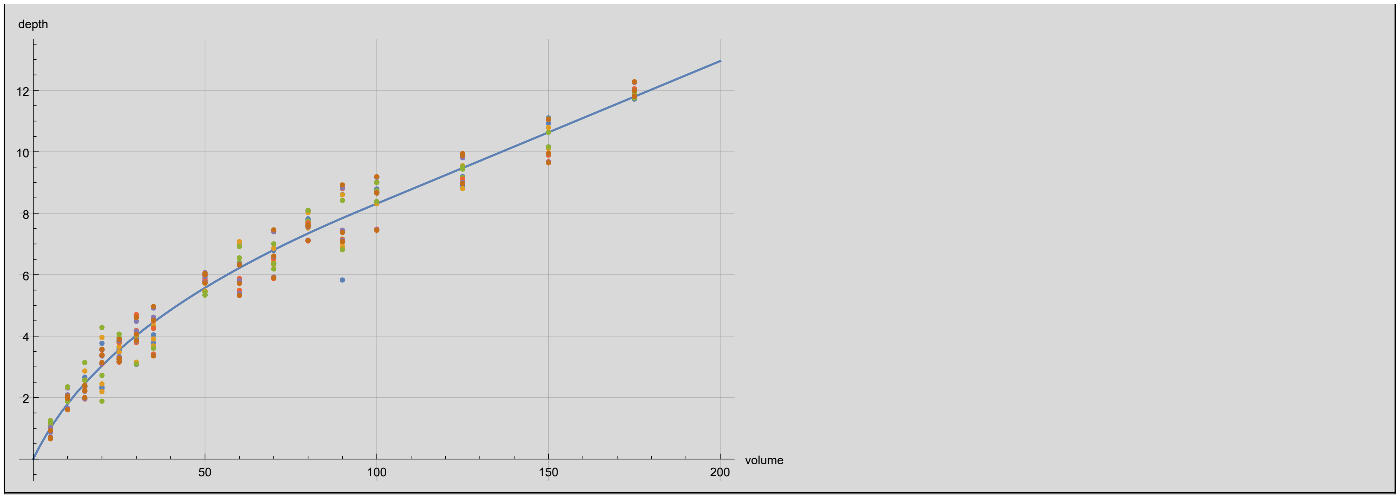
```
toDeg[apexangle[parts[modelledBioRad5][conical]] 2] → 20.2959
```

```
2 rsmall[parts[modelledBioRad5][conical]] → 2.33216
```

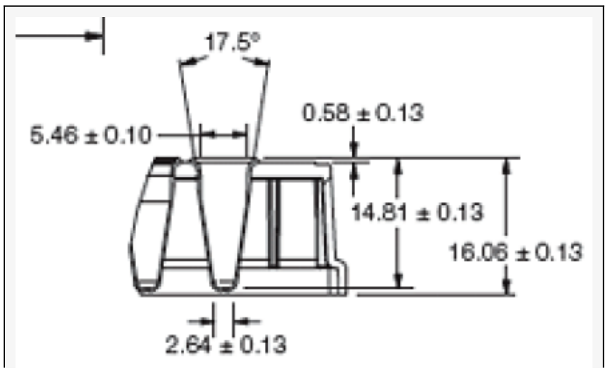
```
2 rbig[parts[modelledBioRad5][conical]] → 5.23718
```

```
volume[modelledBioRad5] → 239.998
```

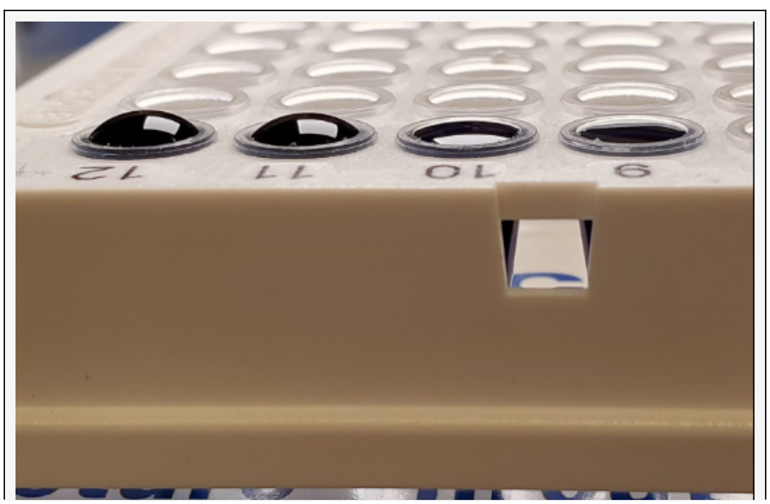
$$\begin{cases} 0 & \text{vol} \le 0 \\ -6.51474 + 2.78018\,(12.8668 + 1.38705\,\text{vol})^{1/3} & \text{vol} \le 95.7748 \\ 8.11502 - 0.046421\,(95.7748 - \text{vol}) & \text{True} \end{cases}$$



While not perfect, the values above compare favorably with the nominal values from the spec:



Also, the predicted to-the-top volume is just shy of 240 μL, which matches experimental data well: In picture, from L to R: 255, 250, 240, and 230 uL

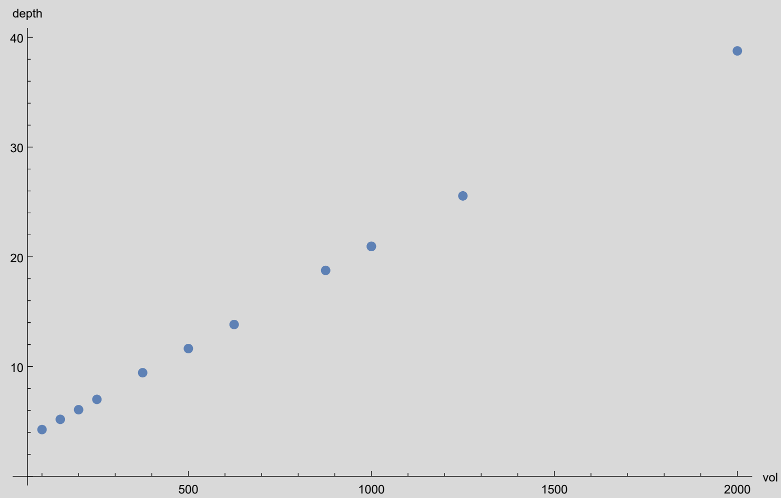


IDT tubes

These are the tubes in which product from IDT DNA is distributed. Correspondence with IDT indicated that these are sourced from Sarsdtedt, part 72.609: https://www.sarstedt.com/en/products/laboratory/screw-cap-micro-tubes-reaction-tubes/screw-cap-micro-tubes/product/72.609/

```
idtData = ArrayReshape[{250, 7.01,  200, 6.07, 150, 5.19, 100, 4.26, 1000,
   20.94, 2000, 38.76, 1000, 20.96, 500, 11.64, 375, 9.44, 625, 13.83, 1250, 25.55, 875, 18.76}, {12, 2}]
ListPlot[idtData, ImageSize → Large, AxesLabel → {"vol", "depth"}, PlotRange → All]
```

```
{{250, 7.01}, {200, 6.07}, {150, 5.19}, {100, 4.26}, {1000, 20.94},
 {2000, 38.76}, {1000, 20.96}, {500, 11.64}, {375, 9.44}, {625, 13.83}, {1250, 25.55}, {875, 18.76}}
```



```
fitIdtData[data_] := Module[{depthFunc, cylinderData, vMin, hMin, offsetCylinderData, hCone, hCyl1,
   hCyl2, hCyl, rCyl, conePart, cylinderPart, errors, err, min, cylinderRules, tube, tubeRules, hOverall, idtRules},
  depthFunc[part_] := Module[{expr, v},
    expr = depthFromVolume[part, v];
    depthFunc[part] = Function[{vol}, expr /. {v → vol}]
   ];
  (* figure out the common radius of the cylinder & cone *)
  cylinderData = Select[data, True &];
  vMin = Min @ cylinderData[[All, 1]];
  hMin = Min @ cylinderData[[All, 2]];
  offsetCylinderData = {#[[1]] - vMin, #[[2]] - hMin} & /@ cylinderData;
  cylinderPart = cylinder[hCyl1, rCyl];
  errors = Function[{vol, depth},
       (depthFunc[cylinderPart][vol] - depth) ^ 2
      ] @@ # & /@ offsetCylinderData;
  err = Total[errors] // N;
  {min, cylinderRules} = NMinimize[{err, assumptions[cylinderPart] }, {hCyl1, rCyl}];
  test @ cylinderRules;

  (* figure out the height of the cone *)
  cylinderPart = cylinder[hCyl2, rCyl];
  conePart = invertedCone[hCone, rCyl];
  tube = conicalTestTube[cylinderPart, conePart, emptyCylinder[]] /. cylinderRules;
  test @ tube;
  errors = Function[{vol, depth},
       (depthFunc[tube][vol] - depth) ^ 2
      ] @@ # & /@ data;
  err = Total[errors] // N;
  {min, tubeRules} = NMinimize[{err }, {hCyl2, hCone}];
  test @ tubeRules;

  (* finally figure out the real height of the cylinder *)
  hOverall = 42; (* from opentrons labware *)
  tube = conicalTestTube[cylinder[hOverall - hCone, rCyl], conePart, emptyCylinder[]] /. cylinderRules /. tubeRules;
  tube
 ]
fittedIdt = fitIdtData[idtData]
test @ volume @ fittedIdt;
```

```
cylinderRules$71417 → {hCyl1$71417 → 6.4908, rCyl$71417 → 4.16389}
```

```
tube$71417 → conicalTestTube[cylinder[hCyl2$71417, 4.16389], invertedCone[hCone$71417, 4.16389], cylinder[0, 0]]
```

```
tubeRules$71417 → {hCyl2$71417 → 1.98558, hCone$71417 → 3.69629}
```

```
conicalTestTube[cylinder[38.3037, 4.16389], invertedCone[3.69629, 4.16389], cylinder[0, 0]]
```

```
volume[fittedIdt] → 2153.47
```

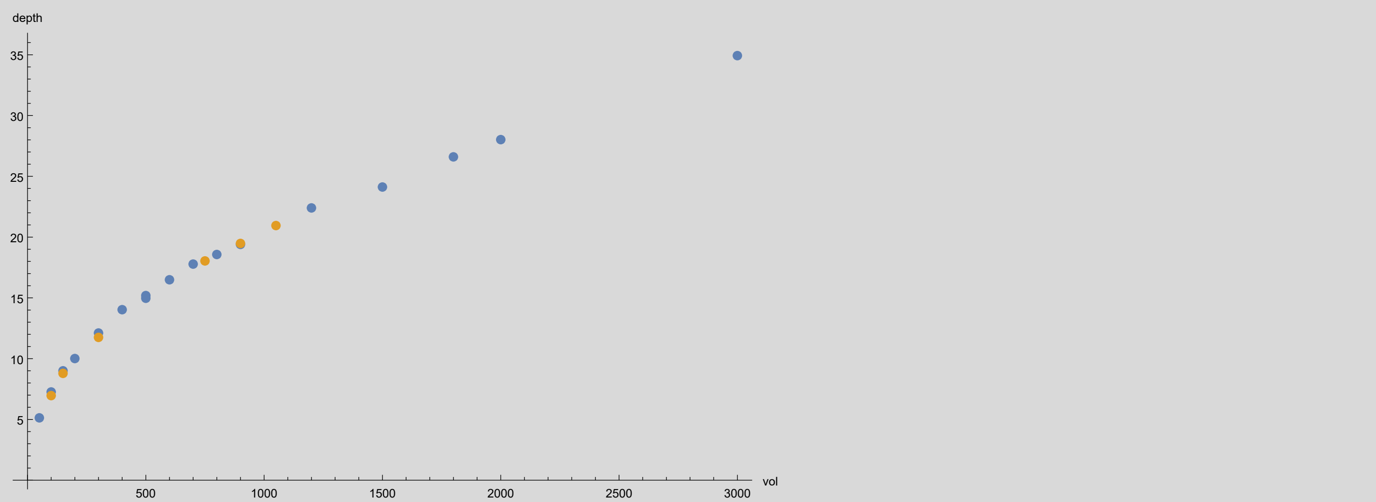
Falcon Tubes

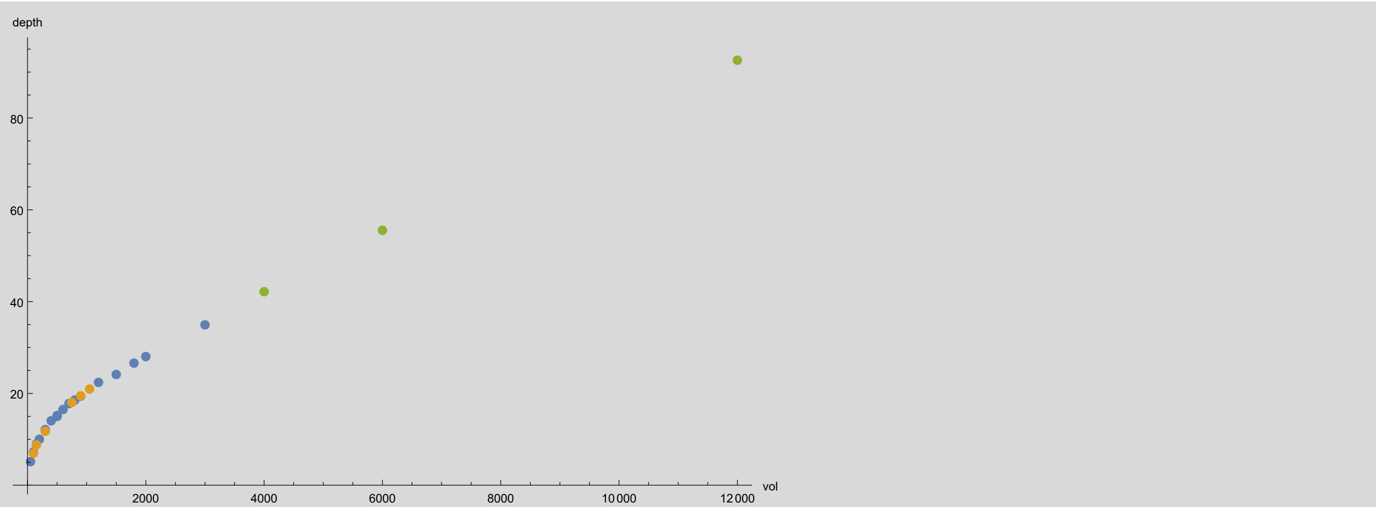
Liquid-volume-to-liquid-depth data was gathered as it was for Eppendorf tubes.

15mL

We have some empirical data for the 15mL Falcon tube.

```
Block[{hBase = 34.93},
  goodFalcon15Data = {
    (*{1000, 19.78},*) {2000, 28.02}, {3000, hBase}, {500, 15.19}, (*{1000, 19.99},*) {50, 5.13}, {100, 7.26},
    {200, 10.01}, {150, 9.00}, {300, 12.11}, {600, 16.49}, {1200, 22.40}, {1800, 26.60},
    {400, 14.03}, {500, 14.97}, {700, 17.78}, {800, 18.57}, {900, 19.40}, {1500, 24.12}
   };
  okFalcon15Data = {
    {100, 6.96}, {150, 8.79}, {300, 11.75}, (*{450, 14.32},*)
    (*{600, 15.89},*) {750, 18.04}, {900, 19.48}, {1050, 20.95}(*, {1200, 20.51}*)
   };
  upperFalcon15Data = {
    {4000, hBase + 7.23}, {6000, hBase + 20.60} , {12000, hBase + 57.66}
   }];
ListPlot[{goodFalcon15Data, okFalcon15Data}, ImageSize → Large, AxesLabel → {"vol", "depth"}, PlotRange → All]
ListPlot[{goodFalcon15Data, okFalcon15Data, upperFalcon15Data}, ImageSize → Large, AxesLabel → {"vol", "depth"}, PlotRange → All]
falcon15Data = Union[goodFalcon15Data ~Join~ okFalcon15Data ~Join~ upperFalcon15Data]
```



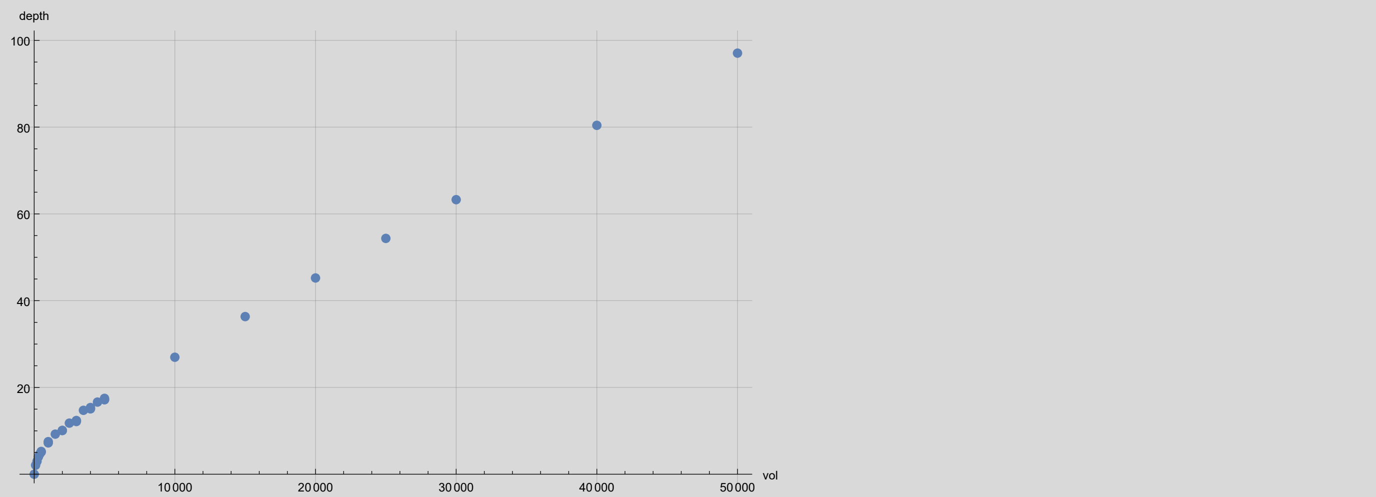


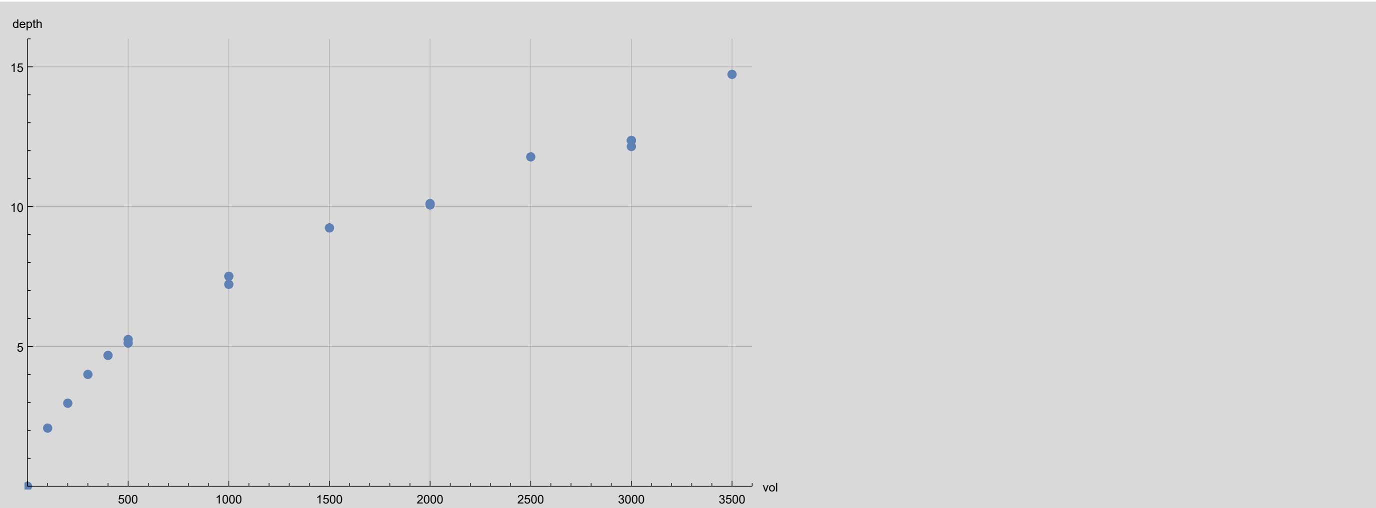

```
{{50, 5.13}, {100, 6.96}, {100, 7.26}, {150, 8.79}, {150, 9.}, {200, 10.01}, {300, 11.75}, {300, 12.11}, {400, 14.03},
 {500, 14.97}, {500, 15.19}, {600, 16.49}, {700, 17.78}, {750, 18.04}, {800, 18.57}, {900, 19.4}, {900, 19.48}, {1050, 20.95},
 {1200, 22.4}, {1500, 24.12}, {1800, 26.6}, {2000, 28.02}, {3000, 34.93}, {4000, 42.16}, {6000, 55.53}, {12000, 92.59}}
```

50mL

https://ecatalog.corning.com/life-sciences/b2c/US/en/Liquid-Handling/Tubes%2 C-Liquid-Handling/Centrifuge-Tubes/Falcon®-Conical-Centrifuge-Tubes/p/352070

```
Block[{},
 falcon50Data = {{0, 0}, {100, 2.08`}, {200, 2.97`}, {300, 4}, {400, 4.68`}, {500, 5.25`}, {500, 5.12`}, {1000, 7.22`}, {1000, 7.51`},
   {1500, 9.24`}, {2000, 10.11`}, {2000, 10.06`}, {2500, 11.78`}, {3000, 12.36`}, {3000, 12.37`}, {3000, 12.15`}, {3500, 14.73`},
   {4000, 15.1`}, {4000, 15.37`}, {4000, 15.07`}, {4500, 16.63`}, {5000, 17.17`}, {5000, 17.49`}, {5000, 17.19`}, {10000, 26.96333333`},
   {15000, 36.33333333`}, {20000, 45.23333333`}, {25000, 54.35333333`}, {30000, 63.28333333`}, {40000, 80.41333333`}, {50000, 97.05333333`}};
 cellPrint @ ListPlot[falcon50Data, ImageSize → Large, AxesLabel → {"vol", "depth"}, PlotRange → All, GridLines → Automatic];
 cellPrint @ ListPlot[falcon50Data, ImageSize → Large, AxesLabel → {"vol", "depth"}, PlotRange → {{0, 3600}, {0, 16}}, GridLines → Automatic];
]
```





Old, no longer true: Unfortunately, this data appears, somehow, to be off; we can't figure out how to fit it reasonably and have anything anywhere close to the apex angle of the cone that we know we need. Update: the issue is that in our world, the 'apex angle' is the *half* angle, not the whole angle. So if the spec says the whole angle is 70 deg, we should be working with 35 degrees. Fixed below...

Analysis

We currently model the Falcon tube with an empty cylinder for the cap. We might want to try our inverted spherical cap, but for now, at least, that doesn't seem worth the effort.

```
Clear[fitFalconData, fitFalcon15Data, fitFalcon50Data]

fitFalcon15Data[data_] := Block[{coneAssumpts, fassumpts, hCone, rmid, hCyl, hTot},
  coneAssumpts = hCone > 15;
  fassumpts = hCone > 18 && hCone < 24.5 && rmid > 6 && hCyl > 75;
  fitFalconData[data, True, 1000, 1200, fassumpts, coneAssumpts, {hTot → 119.46 - 1.39}]
 ]

fitFalcon50Data[data_] := Block[{bottom = 1.88, coneAssumpts, fassumpts, hCone, rmid, hCyl, hTot, αCone, a = 114.55, b = 29.72, c = 27.94,
    d = 16, minWall = 0.97, wallSlop, hConeNominal, hConeSlop = 1, hCylNominal, hCylSlop = 2, rmidNominal, rmidSlop},
   wallSlop = minWall * 2;
   hConeNominal = d - bottom;
   hCylNominal = a - d;
   rmidNominal = (c - 2 wallSlop) / 2;
   rmidSlop = minWall * 1.2; (*1.2. here is very sensitive: indicative of an issue*)
```

```
  coneAssumpts = hConeNominal - hConeSlop < hCone < hConeNominal + hConeSlop && rmidNominal - rmidSlop < rmid < rmidNominal + rmidSlop;
  fassumpts = coneAssumpts && hCylNominal - hCylSlop < hCyl < hCylNominal + hCylSlop;
  test @ coneAssumpts;
  test @ fassumpts;
  fitFalconData[data, False, 3100, 3500, fassumpts, coneAssumpts, {hTot → a - bottom, αCone → toRadian[70 / 2]} ]
 ];

fitFalconData[data_, useCartesianCone_, coneThreshold_, cylThreshold_, fassumpts_, coneAssumpts_, constants_] := Block[
  {threshold, conicalData, cylinderData, conePart, genericDepth, hCone, rmid,
   rbottom, errors, err, min, coneRules, angledCone, cylinderPart, hCyl, rtop, cylinderRules, angledCylinder,
   Δvol, Δh, vMin, hMin, offsetCylinderData, falcon, α, falconRules, first, second, hTot, αCone},

  genericDepth[part_] := Module[{expr, v},
    expr = depthFromVolume[part, v];
    genericDepth[part] = Function[{vol}, expr /. {v → vol}]
   ];

  (* first, fit the cone. this gives us the apex angle and rbottom and rmid *)
  conicalData = Select[data, #[[1]] ≤ coneThreshold &];

  If[useCartesianCone
   ,
   conePart = invertedFrustum[hCone, rmid, rbottom];
   ,
   conePart = invertedFrustum[hCone, rmid, αCone, "apexangle"];
  ];
  conePart = conePart /. constants;

  errors = Function[{vol, depth}, (genericDepth[conePart][vol] - depth) ^2] @@ # & /@ conicalData;
  err = Total[errors] // N;
  {min, coneRules} = NMinimize[{err, assumptions[conePart] && coneAssumpts}, variables[conePart]];
  angledCone = toApexAngled[conePart /. coneRules];
  coneRules = coneRules ~Join~ {rbottom → rsmall[angledCone]};

  (* now for the cylinder. this gives us the apex angle of the cylinder *)
  cylinderData = Select[data, #[[1]] ≥ cylThreshold &]; (* hard to tell for in between data, so we're conservative *)
  vMin = Min @ cylinderData[[All, 1]];
  hMin = Min @ cylinderData[[All, 2]];
  offsetCylinderData = {#[[1]] - vMin, #[[2]] - hMin} & /@ cylinderData;
  cylinderPart = invertedFrustum[hCyl, rtop, rmid] /. coneRules;
  errors = Function[{vol, depth}, (genericDepth[cylinderPart][vol] - depth) ^2] @@ # & /@ offsetCylinderData;
  err = Total[errors] // N;
  {min, cylinderRules} = NMinimize[{err, assumptions[cylinderPart] }, {hCyl, rtop}];
  angledCylinder = toApexAngled[cylinderPart /. cylinderRules];

  falcon = conicalTestTube[
    (invertedFrustum[hCyl, hCyl Tan[α] + rmid, α, "apexangle"] /. {α → apexangle[angledCylinder]}),
    (invertedFrustum[hCone, hCone Tan[α] + rbottom, α, "apexangle"] /. {α → apexangle[angledCone]}),
    emptyCylinder[]
   ];

  hTot = hTot /. constants;
  falcon = falcon /. hCyl → hTot - hCone;
  falcon = falcon /. coneRules;

  errors = Function[{vol, depth},
       (FullSimplify[genericDepth[falcon][vol] - depth, fassumpts]) ^2
      ] @@ # & /@ data;
  err = Total[errors] // N;

  (* put together to get rmid, hCyl, and hCone *)
  second[] := Module[{rule = hCyl → hTot - hCone},
    {min, falconRules} = NMinimize[{err /. rule, fassumpts /. rule }, {hCone, rmid}];
    Function[f, conicalTestTube[
       toCartesian[parts[f]["cylindrical"]],
       toCartesian[parts[f]["conical"]],
       emptyCylinder[]
      ]][falcon /. rule /. falconRules]
   ];
  second[]
 ]
```

```
Clear[fittedFalcon50]
fittedFalcon50 = fitFalcon50Data[falcon50Data];
test @ fittedFalcon50;
test @ volume[fittedFalcon50];
test @ N[2 * toDeg @ apexangle @ parts[fittedFalcon50]["conical"]];
test @ depthFromVolume[fittedFalcon50, volume[fittedFalcon50]];
Block[{expr},
  expr = depthFromVolume[fittedFalcon50, vol];
  cellPrint @ Show[
    Plot[expr, {vol, 0, volume[fittedFalcon50]}, ImageSize → Large, AxesOrigin → {0, 0}],
    ListPlot[falcon50Data]];
  expr = radiusFromDepth[fittedFalcon50, depth];
  cellPrint @ Plot[expr, {depth, 0, height[fittedFalcon50]}, AxesOrigin → {0, 0}]];
```

```
coneAssumpts → 13.12 < hCone < 15.12 && 10.866 < rmid < 13.194
```

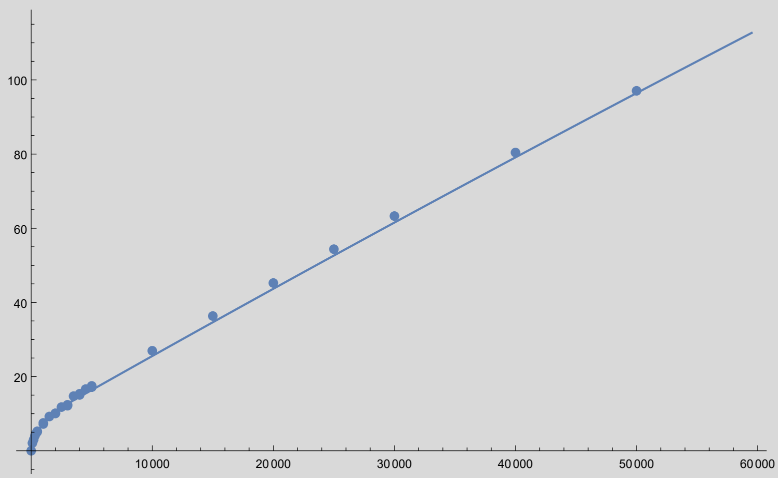
```
fassumpts → 13.12 < hCone < 15.12 && 10.866 < rmid < 13.194 && 96.55 < hCyl < 100.55
```

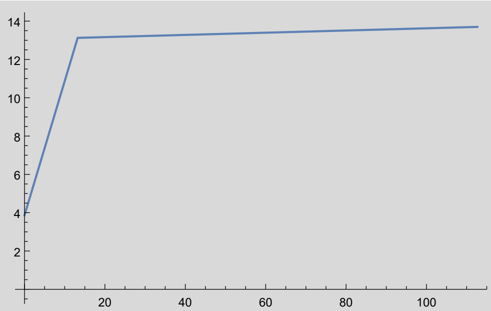
```
fittedFalcon50 → conicalTestTube[invertedFrustum[99.4458, 13.6982, 13.1264], invertedFrustum[13.2242, 13.1264, 3.86673], cylinder[0, 0]]
```

```
volume[fittedFalcon50] → 59 505.8
```

```
N[2 toDeg[apexangle[parts[fittedFalcon50][conical]]]] → 70.
```

```
depthFromVolume[fittedFalcon50, volume[fittedFalcon50]] → 112.67
```



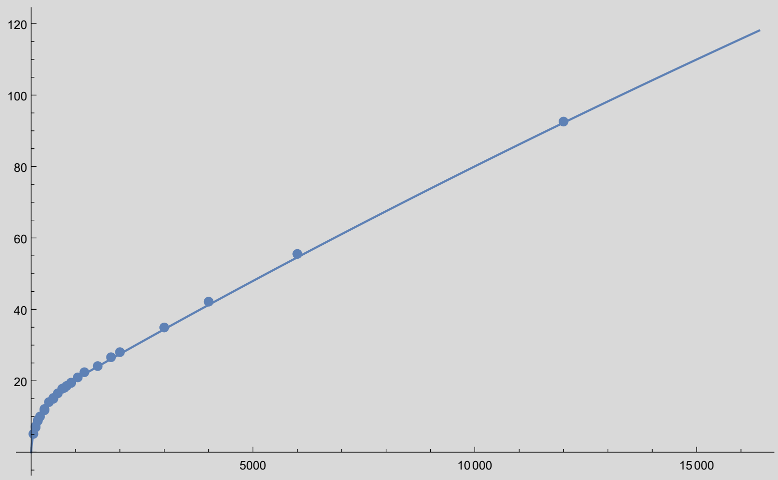


That’s pretty good, but clearly a little on the shy side.

```
fittedFalcon15 = fitFalcon15Data[falcon15Data];
test @ volume[fittedFalcon15];
test @ depthFromVolume[fittedFalcon15, volume[fittedFalcon15]];
Block[{expr},
 expr = depthFromVolume[fittedFalcon15, vol];
 Show[
  Plot[expr, {vol, 0, volume[fittedFalcon15]}, ImageSize → Large, AxesOrigin → {0, 0}],
  ListPlot[falcon15Data]]]
```

```
volume[fittedFalcon15] → 16 410.1
```

```
depthFromVolume[fittedFalcon15, volume[fittedFalcon15]] → 118.07
```



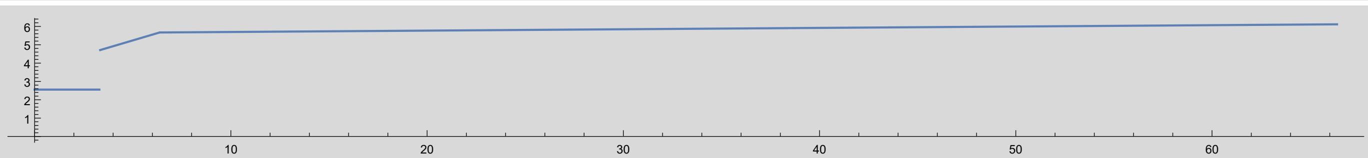
Pipettes and Pipette Tips

Our modelling of pipette tips is less complete than that of tubes. We've completed the work for p50 pipettes with their (usual) 300μL Opentrons tips attached, but work on other models and tips remains incomplete.

p50

```
p50M0 = pipette[
  invertedFrustum[60, 12.23 / 2, 11.34 / 2],
  invertedFrustum[3.05, 11.34 / 2, 9.41 / 2],
  (* what's here isn't actually a cone, but is close enough *)
  cylinder[3.32, 5.11 / 2(*6.91/2*)]] (*forcing monotonicity when tip attached: hack*)
plotProfile[p50M0]
```

```
pipette[invertedFrustum[60, 6.115, 5.67], invertedFrustum[3.05, 5.67, 4.705], cylinder[3.32, 2.555]]
```



10 μL

C:\github\Opentrons\opentrons\shared-data\labware\definitions\2\opentrons_96_tiprack_10ul\1.json

```
opentrons$10μl$tipM0 = pipetteTip[invertedFrustum[39.2, 2.5, 0.75]]
```

```
pipetteTip[invertedFrustum[39.2, 2.5, 0.75]]
```

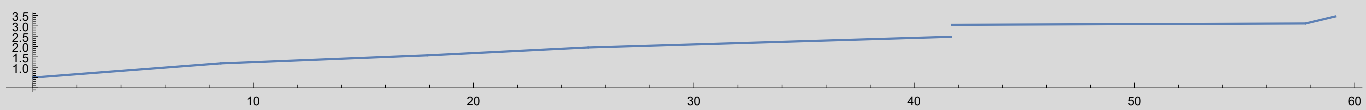
300 μL

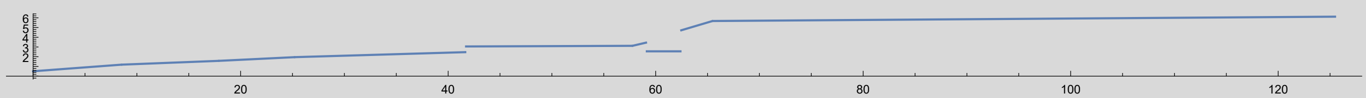
C:\github\Opentrons\opentrons\shared-data\labware\definitions\2\opentrons_96_tiprack_300ul\1.json

```
opentrons$300μl$tipM0 = pipetteTip[invertedFrustum[59.3, 3, 1]];
opentrons$300μl$tipM1 = pipetteTip[
   (* flare *) invertedFrustum[1.35, 6.91 / 2, 6.24 / 2],
   (* ribbed *) invertedFrustum[16.07, 6.24 / 2, 6.11 / 2],
   (* cone section 1 *) invertedFrustum[59.11 - 1.35 - 16.07 - 25.20, 4.94 / 2, 3.91 / 2 ],
   (* cone section 2 *) invertedFrustum[25.20 - 17.90, 3.91 / 2, 3.15 / 2],
   (* cone section 3 *) invertedFrustum[17.90 - 8.53, 3.15 / 2, 2.37 / 2],
   (* cone section 4 *) invertedFrustum[8.53, 2.37 / 2, 1.01 / 2]
  ];
test @ opentrons$300μl$tipM1;
test @ height[opentrons$300μl$tipM1];
plotProfile[opentrons$300μl$tipM1]
plotProfile @ mountTip[p50M0, opentrons$300μl$tipM1]
```

```
opentrons$300μl$tipM1 → pipetteTip[invertedFrustum[1.35, 3.455, 3.12], invertedFrustum[16.07, 3.12, 3.055], invertedFrustum[16.49, 2.47, 1.955],
  invertedFrustum[7.3, 1.955, 1.575], invertedFrustum[9.37, 1.575, 1.185], invertedFrustum[8.53, 1.185, 0.505]]
```

```
height[opentrons$300μl$tipM1] → 59.11
```





Collision Detection: Tips, in Tubes, Moving Laterally

The goal here is for a given combination of pipette, tip, and well to produce a closed form expression that, for any desired depth (from the bottom, aka ‘liquid depth’) gives the available radial clearance before contact with the side wall of the tube is made.

By a good margin, the work in this section was the hardest to develop.

Utilities

```
Clear[findLower, findLowerClause, findUpper, findUpperClause, findBoundSimplify, findClauses, findProcess]
findBoundSimplify[clauses_] := clauses //. {
   Inequality[lower_ , Less, var_ , Less, upper_] :> lower < var && var < upper,
   LessEqual → Less, GreaterEqual → Greater, x_ > y_ :> y < x}
findClauses[clauses_List] := clauses
findClauses[clauses_And] := List @@ clauses
findClauses[other_] := {other}

findProcess[var_, clauses_, op_] := Module[{simplified, list, found},
   simplified = findBoundSimplify[clauses];
   list = Flatten[{simplified /. And → List}];
   found = op[var, #] & /@ list;
   found = Union[Flatten[found]];
   found];

findLower[var_, clauses : (_List | _And)] := findProcess[var, clauses, findLowerClause]
findLower[var_, clause_] := findLower[var, {clause}]
findUpper[var_, clauses : (_List | _And)] := findProcess[var, clauses, findUpperClause]
findUpper[var_, clause_] := findUpper[var, {clause}]
findLowerClause[var_, bound_ < var_] := {bound}
findLowerClause[var_, expr_] := Block[{}, (*printCell["lowerFault" → FullForm[expr]];*) {}]
findUpperClause[var_, var_ < bound_] := {bound}
findUpperClause[var_, expr_] := Block[{}, (*printCell["upperFault" → FullForm[expr]];*) {}]

test @ findLower[z, z <  10];
test @ findUpper[z, z <  10];
test @ findLower[z, z > 10 && z < 11];
test @ findUpper[z, z > 10 && z < 11];
test @ findLower[z, {0.19409486595347666` < z, z < 16.674223638479965`}];
test @ findUpper[z, {0.19409486595347666` < z, z < 16.674223638479965`}];
test @ findLower[z, 0.19409486595347666` < z ≤ 16.674223638479965` ];
test @ findUpper[z, 0.19409486595347666` < z ≤ 16.674223638479965` && z < 17];
```

```
findLower[z, z < 10] → {}
```

```
findUpper[z, z < 10] → {10}
```

```
findLower[z, z > 10 && z < 11] → {10}
```

```
findUpper[z, z > 10 && z < 11] → {11}
```

```
findLower[z, {0.194095 < z, z < 16.6742}] → {0.194095}
```

```
findUpper[z, {0.194095 < z, z < 16.6742}] → {16.6742}
```

```
findLower[z, 0.194095 < z ≤ 16.6742] → {0.194095}
```

```
findUpper[z, 0.194095 < z ≤ 16.6742 && z < 17] → {16.6742, 17}
```

```
Clear[minToPieceWise]

minToPieceWise[expr_, {var_, lower_, upper_}] := Module[{},
  Piecewise[{{expr, Simplify[lower ≤ var && var ≤ upper]}}, Indeterminate]]
minToPieceWise[Min[expr_], {var_, lower_, upper_}] := Module[{},
  Piecewise[{{expr, Simplify[lower ≤ var && var ≤ upper]}}, Indeterminate]]
minToPieceWise[minExpr : Min[_, __], {var_, lower_, upper_}] := Module[{exprs, this, others, and, cond, conds},
   exprs = List @@ minExpr;
   conds = Function[i,
      this = Take[exprs, {i}][[1]];
      others = Drop[exprs, {i}];
      and = Simplify @ And @@ (this ≤ # & /@ others);
      cond = Quiet[Reduce[and && lower ≤ var && var ≤ upper, var], {Reduce::ratnz}];
      {this, cond}
     ] /@ Range[Length[exprs]];
   PiecewiseExpand[Piecewise[conds, Indeterminate]]
  ];

printCell @ minToPieceWise[Min[2.38 + 0.033726 depth, 0.4533 + 0.16904 depth], {depth, 0.19409486595347666`, 16.674223638479965`}];
printCell @ minToPieceWise[7 + depth, {depth, 0.19409486595347666`, 16.674223638479965`}];
printCell @ minToPieceWise[Min[7 + depth], {depth, 0.19409486595347666`, 16.674223638479965`}];
printCell @ minToPieceWise[Min[7 + depth, 4 + 2 depth, 3 + 3 depth], {depth, 0.19409486595347666`, 16.674223638479965`}];
```

$$\begin{cases} 2.38 + 0.033726\,\text{depth} & 14.2387 \le \text{depth} \le 16.6742 \\ 0.4533 + 0.16904\,\text{depth} & 0.194095 \le \text{depth} < 14.2387 \\ \text{Indeterminate} & \text{True} \end{cases}$$

$$\begin{cases} 7 + \text{depth} & 0.194095 \le \text{depth} \le 16.6742 \\ \text{Indeterminate} & \text{True} \end{cases}$$

$$\begin{cases} 7 + \text{depth} & 0.194095 \le \text{depth} \le 16.6742 \\ \text{Indeterminate} & \text{True} \end{cases}$$

$$\begin{cases} 3\,(1 + \text{depth}) & 0.194095 \le \text{depth} < 1. \\ 2\,(2 + \text{depth}) & 1. \le \text{depth} < 3. \\ 7 + \text{depth} & 3. \le \text{depth} \le 16.6742 \\ \text{Indeterminate} & \text{True} \end{cases}$$

```
piecesOf[p_Piecewise] := p[[1]];
trueOf[p_Piecewise] := p[[2]];
condsOf[p_Piecewise] := piecesOf[p][[All, 2]];
exprsOf[p_Piecewise] := piecesOf[p][[All, 1]];
```

```
(* reduces the conditions in a Piecewise *)
reducePiecewise[piecewise_Piecewise, var_] :=
 Piecewise[{#[[1]], Quiet[Reduce[#[[2]], var, Reals], {Reduce::ratnz}]} & /@ piecesOf[piecewise], trueOf[piecewise]]
reducePiecewise[other_, var_] := other
```

```
simplifyPiecewise[piecewise_Piecewise, var_, assumpts_] := Module[{simplify, result},
  simplify[expr_] := FullSimplify[expr, assumpts];
  result = reducePiecewise[piecewise, var];
  result = Piecewise[ { simplify @ #[[1]], #[[2]] } & /@ piecesOf[result], simplify @ trueOf[result]];
  result = PiecewiseExpand[result, assumpts];
  result = result /. {0. → 0, 1. → 1 , -1. → -1};
  result = simplify[result];
  result
 ]
simplifyPiecewise[other_, var_, assumpts_] := Module[{simplify, result},
  simplify[expr_] := FullSimplify[expr, assumpts];
  result = simplify[other];
  result = PiecewiseExpand[result, assumpts];
  result = result /. {0. → 0, 1. → 1 , -1. → -1};
  result = simplify[result];
  result
 ]
```

```
Clear[minValue, maxValue]
maxValue[expr_, range_, assumpts_, var_, default_ : -Infinity] :=
  PiecewiseExpand[-minValue[-expr, range, assumpts, var, -default], assumpts];
minValue[expr_, range_, assumpts_, var_, default_ : Infinity] :=
 Block[{constraints, genExpr, genAssumpts, genRules, genConstraints, genRulesOrder, min, eqn, solns, adjustInfinities},
  adjustInfinities[e_] := e /. (Infinity | DirectedInfinity[1]) → default;
  constraints = range && assumpts;
  {{genExpr, genConstraints}, genRules, genRulesOrder} = genericize[{expr, constraints}, InexactNumberQ];
  min = MinValue[{genExpr, genConstraints}, var];
  min = adjustInfinities[min];
  min = min /. genRules;
  min = PiecewiseExpand[min, assumpts];
  min = FullSimplify[min, assumpts];
  min = adjustInfinities[min]
 ]
maxValue[#[[1]], #[[2]], depth ≥ 0 && depth ≤ 37.8 && z ≥ depth && z ≤ 37.8, z, Indeterminate] &/@ {
   {z, z ≤ 0.194095},
   {z, z ≤ 16.6742 && 8.53 + depth ≥ z}
  } // Column
minValue[z, z ≤ 0 && depth ≥ 0 && depth ≤ 118.07` && z ≥ depth && z ≤ 118.07`, depth ≥ 0 && depth ≤ 118.07` && z ≥ depth && z ≤ 118.07`, z, Indeterminate]
```

```
{ 0.194095       depth ≤ 0.194095
  Indeterminate  True
{ 16.6742        8.1442 ≤ depth ≤ 16.6742
  8.53 + depth   depth < 8.1442
  Indeterminate  True
```

```
{ 0              depth == 0
  Indeterminate  True
```

```
Clear[makeExplicitConditions]
makeExplicitConditions[expr_Piecewise, assumpts_, default_] := Module[{allConds, findTrueCond, trueCond, result},
  allConds[p_Piecewise, ass_] := Simplify[Or @@ condsOf[p], ass];
  findTrueCond[pieces_, ass_] := Simplify[Not[Or @@ (pieces[[All, 2]])], ass];
  findTrueCond[p_Piecewise, ass_] := Simplify[Not[allConds[p, ass]], ass];
  trueCond = findTrueCond[expr, assumpts];
  result = Piecewise[piecesOf[expr] ~Join~ {{trueOf[expr], trueCond}}, Indeterminate];
  result]
```

Main Event

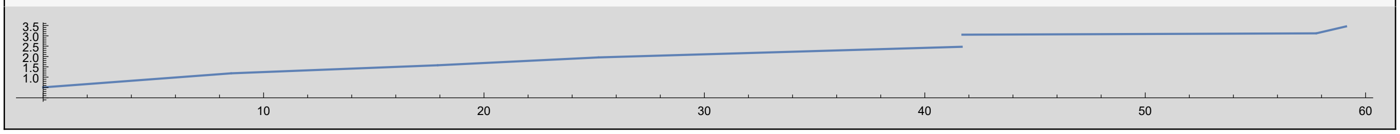
```
Clear[minClearanceFromDepth]

minClearanceFromDepth[tube_, tip_, depth_?NumericQ] := Module[{tubeHeight, assumpts, expr, z},
  tubeHeight = height[tube];
  assumpts = assumptions[tube] && assumptions[tip] && depth ≥ 0 && depth ≤ tubeHeight && z ≥ depth;
  expr = radiusFromDepth[tube, z] - outsideRadiusFromDepth[tip, z - depth];
  expr = PiecewiseExpand[expr, assumpts, Reals];
  (*printCell[Plot[expr, {z, depth, tubeHeight}, AxesLabel→{"z", "clearance"}, AxesOrigin→{0,0},
     GridLines→Automatic, PlotLabel→StringForm["Clearance as function of z with depth=``",depth]]];*)
  FullSimplify[MinValue[{expr, depth ≤ z && z ≤ tubeHeight}, z], assumpts]
 ]

minClearanceFromDepth[tube_, tip_, depth_Symbol] := Block[{tubeHeight, expr, zDepthAssumpts,
   tubeTipAssumpts, assumpts, z, newConds, plotRegion, lowers, uppers, applyZ, allSolns, extremas, mins, min, bound},
  plotRegion[region_, upper_: tubeHeight] := RegionPlot[region, {depth, 0, upper},
    {z, 0, upper}, ImageSize → 150, BoundaryStyle → Thick, GridLines → Automatic];

  tubeHeight = height[tube];
  tubeTipAssumpts = assumptions[tube] && assumptions[tip];
  zDepthAssumpts = depth ≥ 0 && depth ≤ tubeHeight && z ≥ depth && z ≤ tubeHeight;
  assumpts = tubeTipAssumpts && zDepthAssumpts;

  (* our fundamental clearance expression is the difference in the radii. The pipette tip is above the bottom of the tube by 'depth' *)
  expr = radiusFromDepth[tube, z] - outsideRadiusFromDepth[tip, z - depth];

  (* simplify *)
  expr = simplifyPiecewise[expr, depth, tubeTipAssumpts && zDepthAssumpts];

  (* Make all conditions explicit rather than implicit *)
  expr = makeExplicitConditions[expr, tubeTipAssumpts && zDepthAssumpts, Indeterminate];

  (* Manifest z ≥ depth etc in the conditions *)
```

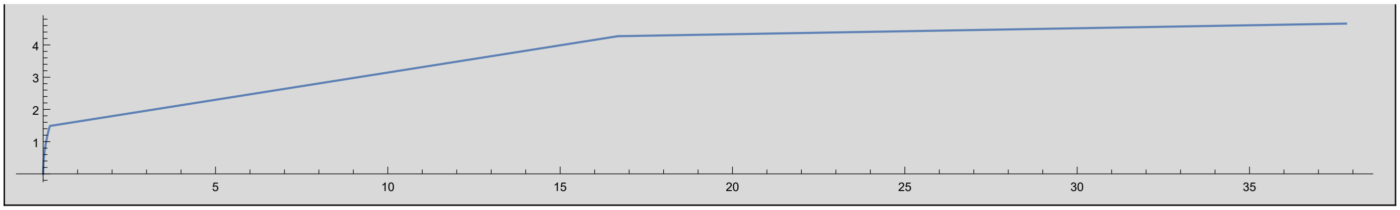
```
newConds = # && zDepthAssumpts &/@ condsOf[expr];
expr = Piecewise[Transpose[{exprsOf @ expr, newConds}], trueOf[expr]];

(*cellPrint @ Row[{(*plotRegion[condsOf[expr][[2]], 0.2],*)Row[plotRegion /@ condsOf[expr]]}];*)

(* figure out lower and upper bounds for z in each of the pieces. 'Indeterminiate' helps nuke Complex[], Infinity, etc *)
lowers  = minValue[z, #, assumpts, z, Indeterminate] & /@ condsOf[expr];
uppers = maxValue[z, #, assumpts, z, Indeterminate] & /@ condsOf[expr];

(* pair those with the corresponding expressions *)
lowers = pairUp[exprsOf[expr], lowers];
uppers = pairUp[exprsOf[expr], uppers];

(* apply those expressions at the lower and upper bounds *)
applyZ[zExpr_, HoldPattern @ Piecewise[pieces_, true_]] :=
 Piecewise[{Simplify[zExpr /. z → #[[1]]], #[[2]]} & /@ pieces, Simplify[ zExpr /. z → true]];
applyZ[zExpr_, other_] := Simplify[zExpr /. z → other];
lowers = applyZ[#[[1]], #[[2]]] & /@ lowers;
uppers = applyZ[#[[1]], #[[2]]] & /@ uppers;

(* figure out if there are any extrema on the interior of the various regions *)
allSolns = Solve[D[#, z] == 0, z] & /@ exprsOf[expr];
extremas = Function[{solns, e, cond}, Module[{result},
       result = (Function[soln,
            If [Simplify[cond /. soln] == False, {}, Piecewise[{{e /. soln, cond /. soln}}, Indeterminate]]
           ] /@ solns);
       Flatten[result]
      ]
     ] @@ # & /@ pairUp[allSolns, exprsOf[expr], condsOf[expr]];

(* put lower and upper together with extremas and then Min over each piece *)
mins = pairUp[lowers, uppers];
mins = Flatten[mins, {1}];
mins = Flatten /@ mins;
mins = (Min /@ mins);

(* Infinity is friendlier since we're using Min *)
mins = mins /. {Indeterminate → Infinity};

(* Simplify each piece *)
mins = PiecewiseExpand[#, True] & /@ mins;
mins = FullSimplify[#, True] & /@ mins;
mins = simplifyPiecewise[#, depth, True] & /@mins;

(* min across the pieces *)
min = Min @@ mins;
min = PiecewiseExpand[min, True, Reals];

(* simplify *)
min = simplifyPiecewise[min, depth, True];
min = FullSimplify[min, zDepthAssumpts];

newConds = # && depth ≥ 0 && depth ≤ tubeHeight &/@ condsOf[min];
min = Piecewise[Transpose[{exprsOf @ min, newConds}], trueOf[min]];
min = simplifyPiecewise[min, depth, depth ≥ 0 && depth ≤ tubeHeight];

(* tidy up with explicit conditions *)
min = makeExplicitConditions[min, True, Indeterminate];

(* clamp to tubeHight above *)
min = Piecewise[{{Infinity, depth > tubeHeight}} ~Join~ piecesOf[min] ~Join~ {}, Indeterminate];
min = simplifyPiecewise[min, depth, True];

(* clamp to zero below *)
min = Piecewise[pairUp[Max[0, #] &/@ exprsOf[min], condsOf[min]], trueOf[min]];
min = simplifyPiecewise[min, depth, True];
min = simplifyPiecewise[min, depth, True];
min = makeExplicitConditions[min, True, Indeterminate];
min = FullSimplify[min, depth ≥ 0];

(* sort in an order convenient for code *)
bound[piece_] := ((Max @@ findUpper[depth, piece[[2]]]) /. -Infinity → Infinity);
min = Piecewise[Sort[piecesOf[min], bound[#1] < bound[#2] &], trueOf[min]];
```

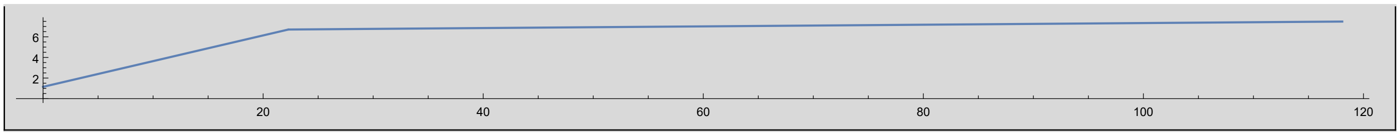
```
  min
 ]
```

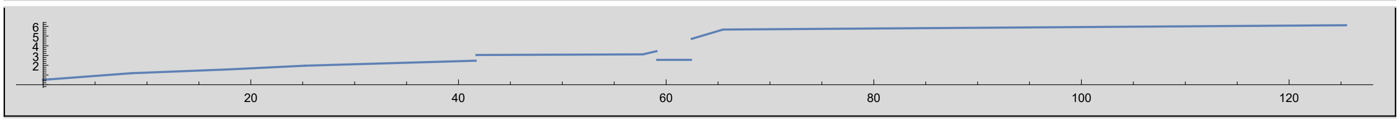
Tests

```
plotProfile @ opentrons$300μl$tipM1
plotProfile @ fittedEppendorf1$5M1
plotProfile @ fittedFalcon15
plotProfile @ mountTip[p50M0, opentrons$300μl$tipM1]
```









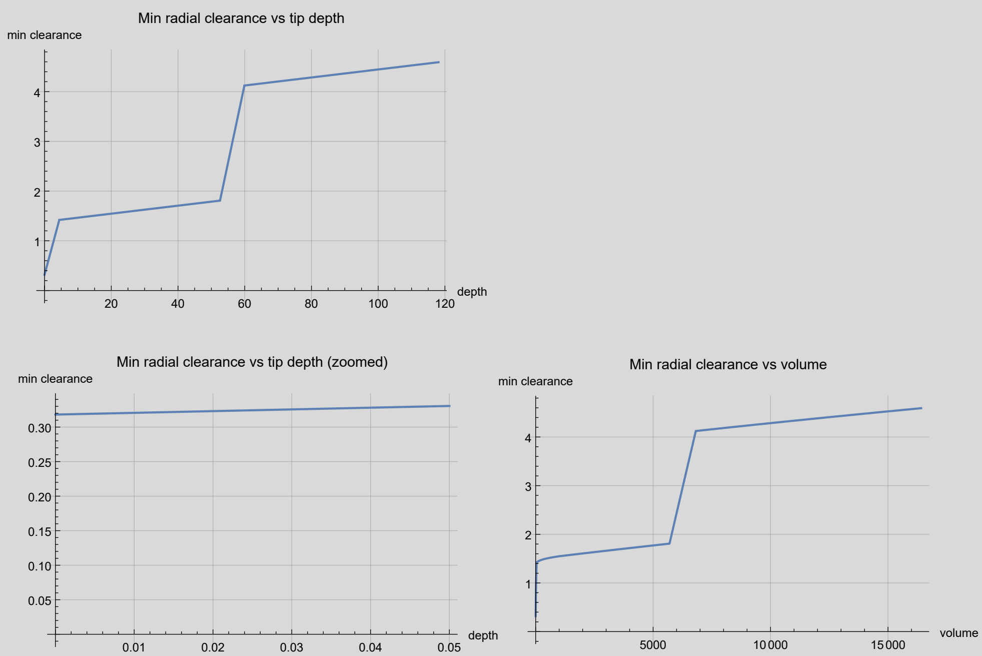
fittedFalcon15

The value for identically zero isn’t correct (it should be as in 0 <= depth < 4.21826), but good enough for us in our needs (we’ll adjust when we pythonize).

```
testResult = minClearanceFromDepth[fittedFalcon15, mountTip[p50M0, opentrons$300μl$tipM1], depth]
```

```
⎧ 0.318101 + 0.249291 depth    0 < depth < 4.42012
⎪ 1.38438 + 0.00805941 depth   4.42012 ≤ depth < 52.59
⎨ -14.8309 + 0.316393 depth    52.59 ≤ depth < 59.9064
⎪ 3.64027 + 0.00805941 depth   59.9064 ≤ depth ≤ 118.07
⎪ ∞                            depth > 118.07
⎩ 0                            True
```

```
testResult = minClearanceFromDepth[fittedFalcon15, mountTip[p50M0, opentrons$300µl$tipM1], depth];
plotfunc[volume_] := depthFromVolume[fittedFalcon15, volume]
Row @ {
  Plot[testResult, {depth, 0, 118.07}, AxesLabel → {"depth", "min clearance"},
   PlotLabel → "Min radial clearance vs tip depth", GridLines → Automatic, AxesOrigin → {0, 0}, ImageSize → Medium],
  Plot[testResult, {depth, 0, 0.05}, AxesLabel → {"depth", "min clearance"},
   PlotLabel → "Min radial clearance vs tip depth (zoomed)", GridLines → Automatic, AxesOrigin → {0, 0}, ImageSize → Medium],
  Plot[testResult /. depth → plotfunc[vol], {vol, 0, volume[fittedFalcon15]}, AxesLabel → {"volume", "min clearance"},
   PlotLabel → "Min radial clearance vs volume", GridLines → Automatic, AxesOrigin → {0, 0}, ImageSize → Medium]
 }
```

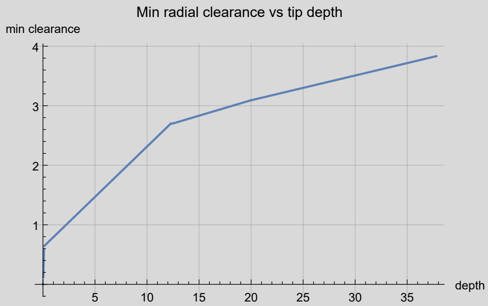


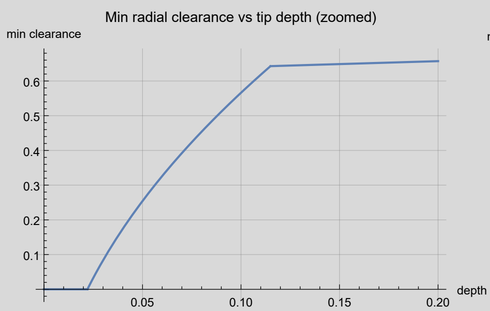
fittedEppendorf1$5M1

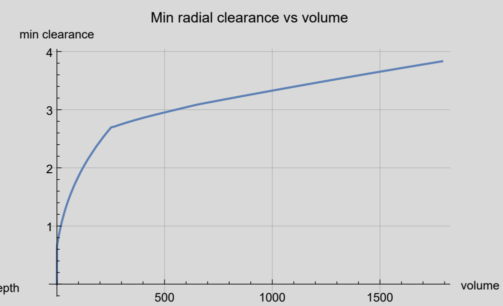
```
testResult = minClearanceFromDepth[fittedEppendorf1$5M1, mountTip[p50M0, opentrons$300µl$tipM1], depth]
```

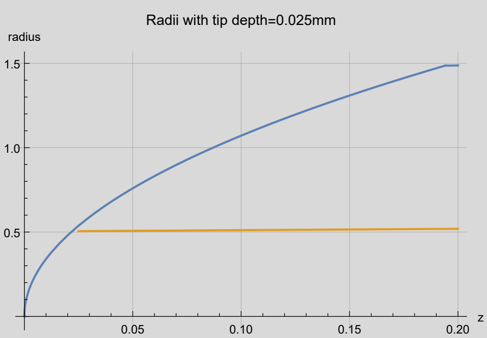
$$\begin{cases} -0.505 + 2.26986\sqrt{(2.24622 - 0.194089\,\text{depth})\,\text{depth}} & 0.022078 < \text{depth} < 0.114975 \\ 0.623346 + 0.169045\,\text{depth} & 0.114975 \le \text{depth} < 12.2688 \\ 2.31416 + 0.031231\,\text{depth} & 12.2688 \le \text{depth} < 12.6 \\ 2.05178 + 0.0520548\,\text{depth} & 12.6 \le \text{depth} < 19.9 \\ 2.25939 + 0.0416222\,\text{depth} & 19.9 \le \text{depth} \le 37.8 \\ \infty & \text{depth} > 37.8 \\ 0 & \text{True} \end{cases}$$

```
testResult = minClearanceFromDepth[fittedEppendorf1$5M1, mountTip[p50M0, opentrons$300μl$tipM1], depth];
plotfunc[volume_] := depthFromVolume[fittedEppendorf1$5M1, volume]
Row @ {
  Plot[testResult, {depth, 0, 37.8}, AxesLabel → {"depth", "min clearance"},
   PlotLabel → "Min radial clearance vs tip depth", GridLines → Automatic, AxesOrigin → {0, 0}, ImageSize → Medium],
  Plot[testResult, {depth, 0, 0.2}, AxesLabel → {"depth", "min clearance"}, PlotLabel → "Min radial clearance vs tip depth (zoomed)",
   GridLines → Automatic, AxesOrigin → {0, 0}, ImageSize → Medium],
  Plot[testResult /. depth → plotfunc[vol], {vol, 0, volume[fittedEppendorf1$5M1]}, AxesLabel → {"volume", "min clearance"},
   PlotLabel → "Min radial clearance vs volume", GridLines → Automatic, AxesOrigin → {0, 0}, ImageSize → Medium]
 }
Row @ {Plot[{radiusFromDepth[fittedEppendorf1$5M1, z], outsideRadiusFromDepth[mountTip[p50M0, opentrons$300μl$tipM1], z - 0.025]},
   {z, 0, 0.2}, AxesLabel → {"z", "radius"}, PlotLabel → "Radii with tip depth=0.025mm", GridLines → Automatic, ImageSize → Medium]
 }
```









Python-izable Function Creation

In this section, we exhibit the detailed definition of the functions which can be copied into Python form with the minimal amount of human editing (in particular, numeric constants and expressions can be literally copied and pasted).

Utilities

A simple utility helps us create the text which can be copied and pasted.

```
Clear[cFormat, cubeRoot, square, cube, sqrt]
cFormat[p_Piecewise] := Module[{pieces, default, formatted, op, rules},
   pieces = p[[1]];
   default = p[[2]];
   rules = {
     x_^(1/3) :→ cubeRoot[x],
     x_^(-1/3) :→ 1 / cubeRoot[x],
     x_^2 :→ square[x],
     x_^3 :→ cube[x],
     Sqrt[x_] :→ sqrt[x]
    };
   op = Function[{expr},
     CForm[expr //. rules]
    ];
   formatted = {op[#[[1]]], #[[2]]} &/@ pieces;
   Piecewise[formatted, op[default]]];
```

Tubes

For each of the tubes we care about, here we select the model for same that we find most accurate.

```
(tubes = {
      falcon15ml → fittedFalcon15,
      falcon50ml → fittedFalcon50,
      eppendorf1$5ml → fittedEppendorf1$5M1,
      eppendorf5$0ml → fittedEppendorf5$0M0,
      idtTube → fittedIdt,
      bioradPlateWell → (*modelBioRad3[]*) modelledBioRad5
      (*, generic → toCanonical @ conicalTestTube[{idTop, idHip, idBottom}, {hTop, hBottom}]*)
     } // Association) // Normal // ColumnForm
```

```
falcon15ml → conicalTestTube[invertedFrustum[95.7737, 7.47822, 6.70634], invertedFrustum[22.2963, 6.70634, 1.14806], cylinder[0, 0]]
falcon50ml → conicalTestTube[invertedFrustum[99.4458, 13.6982, 13.1264], invertedFrustum[13.2242, 13.1264, 3.86673], cylinder[0, 0]]
eppendorf1$5ml → conicalTestTube[invertedFrustum[21.1258, 4.66267, 4.272], invertedFrustum[16.4801, 4.272, 1.48612], invertedSphericalCap[0.194089
eppendorf5$0ml → conicalTestTube[invertedFrustum[35.8967, 7.08628, 6.37479], invertedFrustum[18.3424, 6.37479, 1.50899], invertedSphericalCap[1.16
idtTube → conicalTestTube[cylinder[38.3037, 4.16389], invertedCone[3.69629, 4.16389], cylinder[0, 0]]
bioradPlateWell → conicalTestTube[cylinder[6.69498, 2.61859], invertedFrustum[8.11502, 2.61859, 1.16608], cylinder[0, 0]]
```

A helper function is necessary to create the output.

```
Clear[printAndPlot]
printAndPlot[name_] := Block[{simplify, expr, tube, h},
  simplify[fn_] := FullSimplify[fn, assumptions[tube]];

  CellPrint[TextCell[name, "Subsubsection"]];
  tube = tubes[name];
  test @ parts[tube];

  If[ToString[name] == "generic",
   test @ simplify @ volume[tube];
   test @ simplify @ depthFromVolume[tube, vol];
   test @ simplify @ volumeFromDepth[tube, depth];
   test @ simplify @ radiusFromDepth[tube, depth];
   ,
   test @ N @ volume[tube];
   test @ N @ volumeFromDepth[tube, height[tube]];
   test @ N @ height[tube];
   test @ N @ depthFromVolume[tube, volume[tube]];
   test @ N @ (2 * radiusFromDepth[tube, height[tube]]);

   test @  N @ simplify @ depthFromVolume[tube, vol];
   test @ cFormat @ simplify @ depthFromVolume[tube, vol];
   expr = N @ depthFromVolume[tube, vol];
   printCell @ Plot[expr, {vol, 0, volume[tube]}, AxesLabel → {"volume", "depth"},
     PlotLabel → ToString[name] <> ": depth from volume", AxesOrigin → {0, 0}, ImageSize → Medium];

   test @ N @ simplify @ volumeFromDepth[tube, depth];
   test @ cFormat @ simplify @ volumeFromDepth[tube, depth];
   expr = N @ volumeFromDepth[tube, depth];
   printCell @ Plot[expr, {depth, 0, height[tube]}, AxesLabel → {"depth", "volume"},
     PlotLabel → ToString[name] <> ": volume from depth", AxesOrigin → {0, 0}, ImageSize → Medium];

   test @ N @ simplify @ radiusFromDepth[tube, depth];
   test @ cFormat @ simplify @ radiusFromDepth[tube, depth];
   expr = N @ radiusFromDepth[tube, depth];
   printCell @ Plot[expr, {depth, 0, height[tube]}, AxesLabel → {"depth", "radius"},
     PlotLabel → ToString[name] <> ": radius from depth", AxesOrigin → {0, 0}, ImageSize → Medium];
  ]]
printAndPlot /@ Keys[tubes];
```

falcon15ml

```
parts[tube] →
 ⟨|cylindrical → invertedFrustum[95.7737, 7.47822, 6.70634], conical → invertedFrustum[22.2963, 6.70634, 1.14806], cap → cylinder[0, 0]|⟩
```

```
N[volume[tube]] → 16410.1
```

```
N[volumeFromDepth[tube, height[tube]]] → 16410.1
```

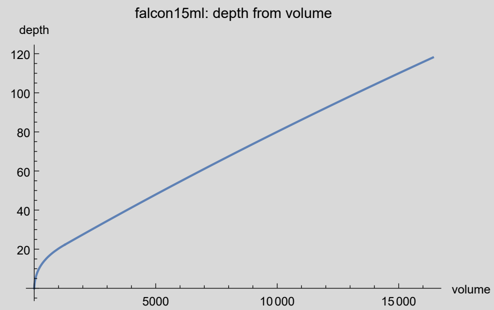
```
N[height[tube]] → 118.07
```

```
N[depthFromVolume[tube, volume[tube]]] → 118.07
```

```
N[2 radiusFromDepth[tube, height[tube]]] → 14.9564
```

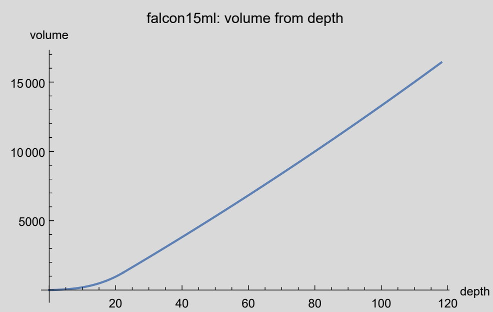
N[simplify[depthFromVolume[tube, vol]]] → $\begin{cases} 0. & vol \le 0. \\ -4.60531 + 1.42522\ (33.739 + 5.30776\ vol)^{1/3} & vol \le 1260.65 \\ -809.817 + 27.1195\ (27957.8 + 0.737091\ vol)^{1/3} & True \end{cases}$

```
cFormat[simplify[depthFromVolume[tube, vol]]] →
 0                                                                                      vol ≤ 0
 -4.605312927271903 + 1.425220154402649*cubeRoot(33.73895064080807 + 5.3077630053562075*vol)    vol ≤ 1260.65
 -809.8165210055173 + 27.119471721476614*cubeRoot(27957.824136197134 + 0.7370907258662586*vol)  True
```



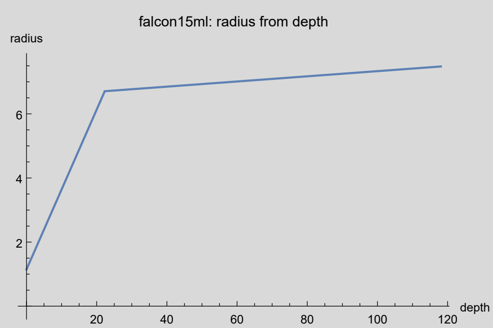

```
N[simplify[volumeFromDepth[tube, depth]]] → ⎧ depth (4.14078 + (0.899131 + 0.0650793 depth) depth)                 0. < depth ≤ 22.2963
                                            ⎨ -1806.01 + depth (133.823 + (0.165251 + 0.0000680198 depth) depth)   depth > 22.2963
                                            ⎩ 0.                                                                   True
```

```
cFormat[simplify[volumeFromDepth[tube, depth]]] →
 ⎧ depth*(4.1407799998941535 + (0.8991310830091779 + 0.06507926078773585*depth)*depth)                              0 < depth ≤ 22.2963
 ⎨ -1806.0097363707396 + depth*(133.82273354274736 + (0.1652506834221966 + 0.00006801980413086301*depth)*depth)   depth > 22.2963
 ⎩ 0                                                                                                                 True
```




```
N[simplify[radiusFromDepth[tube, depth]]] → ⎧ 0.                         depth ≤ 0.
                                            ⎨ 1.14806 + 0.249291 depth   depth ≤ 22.2963
                                            ⎩ 6.52665 + 0.00805941 depth True
```

```
cFormat[simplify[radiusFromDepth[tube, depth]]] → ⎧ 0                                          depth ≤ 0
                                                  ⎨ 1.1480641142716852 + 0.2492912278496944*depth   depth ≤ 22.2963
                                                  ⎩ 6.526645316147934 + 0.008059412406212692*depth  True
```



falcon50ml

```
parts[tube] →
 ⟨|cylindrical → invertedFrustum[99.4458, 13.6982, 13.1264], conical → invertedFrustum[13.2242, 13.1264, 3.86673], cap → cylinder[0, 0]|⟩
```

```
N[volume[tube]] → 59505.8
```

```
N[volumeFromDepth[tube, height[tube]]] → 59505.8
```

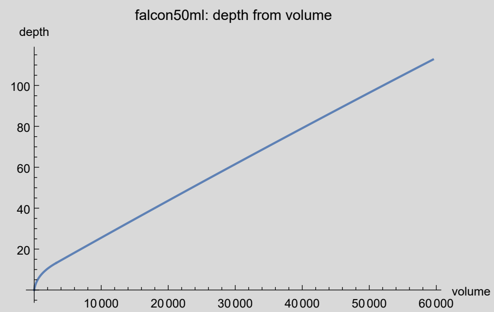
```
N[height[tube]] → 112.67
```

```
N[depthFromVolume[tube, volume[tube]]] → 112.67
```

```
N[2 radiusFromDepth[tube, height[tube]]] → 27.3965
```

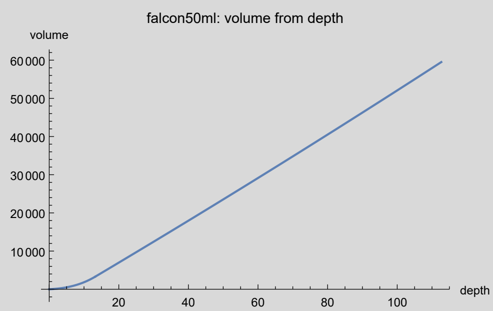
```
N[simplify[depthFromVolume[tube, vol]]] → { 0.                                              vol ≤ 0.
                                            -5.52226 + 0.603925 (764.544 + 8.84237 vol)^(1/3)   vol ≤ 3296.08
                                            -2269.69 + 37.5388 (223120. + 0.546029 vol)^(1/3)   True
```

```
cFormat[simplify[depthFromVolume[tube, vol]]] →
 { 0                                                                                  vol ≤ 0
   -5.522264395071952 + 0.6039249881108911*cubeRoot(764.5441851977812 + 8.842372775534407*vol)     vol ≤ 3296.08
   -2269.6881765411304 + 37.538777353484434*cubeRoot(223119.88753911393 + 0.5460286683567588*vol)  True
```



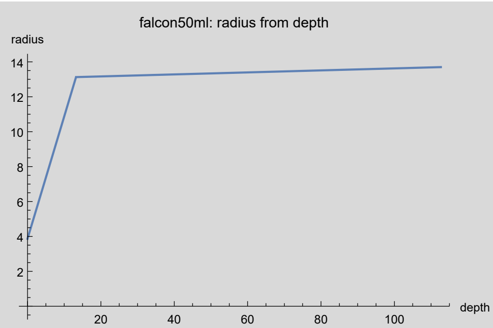

```
N[simplify[volumeFromDepth[tube, depth]]] → { depth (46.9719 + (8.50591 + 0.513431 depth) depth)                      0. < depth ≤ 13.2242
                                              -3820.91 + depth (535.054 + (0.235739 + 0.0000346214 depth) depth)      depth > 13.2242
                                              0.                                                                      True
```

```
cFormat[simplify[volumeFromDepth[tube, depth]]] →
 { depth*(46.97186764441949 + (8.505907048988277 + 0.5134311120983222*depth)*depth)                                0 < depth ≤ 13.2242
   -3820.9148917040493 + depth*(535.0542643832791 + (0.23573910721016753 + 0.000034621364826925 24*depth)*depth)    depth > 13.2242
   0                                                                                                                True
```




```
N[simplify[radiusFromDepth[tube, depth]]] → { 0.                          depth ≤ 0.
                                              3.86673 + 0.700208 depth    depth ≤ 13.2242
                                              13.0504 + 0.00574987 depth  True
```

```
cFormat[simplify[radiusFromDepth[tube, depth]]] → { 0                                             depth ≤ 0
                                                    3.8667311574164636 + 0.7002075382097096*depth  depth ≤ 13.2242
                                                    13.050404667978436 + 0.0057498667891316*depth  True
```



eppendorf1$5ml

```
parts[tube] → ⟨|cylindrical → invertedFrustum[21.1258, 4.66267, 4.272],
   conical → invertedFrustum[16.4801, 4.272, 1.48612], cap → invertedSphericalCap[0.194089, 1.48612, rCap]|⟩
```

```
N[volume[tube]] → 1788.68
```

```
N[volumeFromDepth[tube, height[tube]]] → 1788.68
```

```
N[height[tube]] → 37.8
```

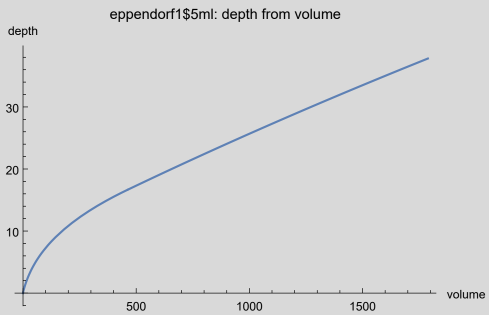
```
N[depthFromVolume[tube, volume[tube]]] → 37.8
```

```
N[2 radiusFromDepth[tube, height[tube]]] → 9.32533
```

N[simplify[depthFromVolume[tube, vol]]] →

$$\begin{cases} \frac{3.23462}{\left(-3.\,\text{vol}+\sqrt{106.321+9.\,\text{vol}^2}\right)^{1/3}} - 0.682784 \left(-3.\,\text{vol}+\sqrt{106.321+9.\,\text{vol}^2}\right)^{1/3} & \text{vol} \le 0.677156 \\ -8.59717 + 2.32458\,(52.2891 + 2.66032\,\text{vol})^{1/3} & \text{vol} \le 463.316 \\ -214.342 + 19.5617\,(1474.21 + 0.373056\,\text{vol})^{1/3} & \text{True} \end{cases}$$

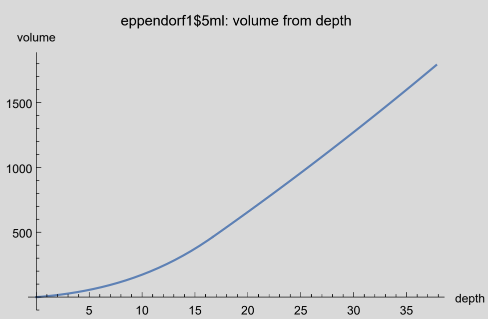
```
cFormat[simplify[depthFromVolume[tube, vol]]] →
  3.2346219418580273/cubeRoot(-3*vol + sqrt(106.32134388676978 +                            vol ≤ 0.677156
    9*square(vol))) - 0.6827840632552957*cubeRoot(-3*vol + sqrt(106.32134388676978 + 9*square(vol)))
  -8.597167565068995 + 2.324576725605449*cubeRoot(52.28906291516273 + 2.6603249808253*vol)     vol ≤ 463.316
  -214.34185528911152 + 19.561687003351448*cubeRoot(1474.2109284979651 + 0.373055557325541*vol)    True
```



N[simplify[volumeFromDepth[tube, depth]]] →

$$\begin{cases} 3.46918\,\text{depth} + 0.523599\,\text{depth}^3 & \text{depth} \le 0.194089 \\ -0.639988 + \text{depth}\,(6.63538 + (0.77181 + 0.029925\,\text{depth})\,\text{depth}) & 0.194089 < \text{depth} \le 16.6742 \\ -425.344 + \text{depth}\,(49.3563 + (0.230269 + 0.000358103\,\text{depth})\,\text{depth}) & \text{True} \end{cases}$$

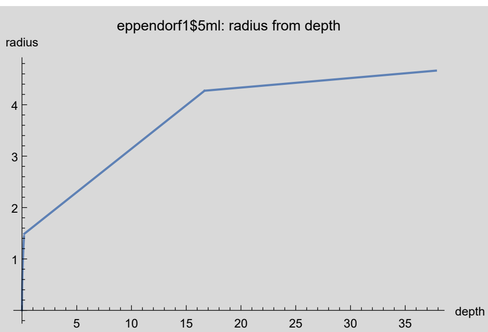
```
cFormat[simplify[volumeFromDepth[tube, depth]]] →
  3.4691795769129103*depth + 0.5235987755982988*cube(depth)                                         depth ≤ 0.194089
  -0.6399880172049095 + depth*(6.635378322870171 + (0.7718098167389763 + 0.029924965049919598*depth)*depth)   0.194089 < depth ≤ 16.6742
  -425.3442649166699 + depth*(49.356322662997606 + (0.2302691772282374 + 0.0003581026781068067*depth)*depth)   True
```



N[simplify[radiusFromDepth[tube, depth]]] →

$$\begin{cases} 2.26986\sqrt{(2.24622 - 0.194089\,\text{depth})\,\text{depth}} & \text{depth} \le 0.194089 \\ 1.45331 + 0.169045\,\text{depth} & \text{depth} \le 16.6742 \\ 3.96366 + 0.0184922\,\text{depth} & \text{True} \end{cases}$$

```
cFormat[simplify[radiusFromDepth[tube, depth]]] →
  2.2698647709367252*sqrt((2.2462186978285 - 0.19408860160231214*depth)*depth)   depth ≤ 0.194089
  1.453308817402276 + 0.16904507285715673*depth                                   depth ≤ 16.6742
  3.9636606122761364 + 0.018492238050892146*depth                                 True
```



eppendorf5$0ml

```
parts[tube] → ⟨|cylindrical → invertedFrustum[35.8967, 7.08628, 6.37479],
  conical → invertedFrustum[18.3424, 6.37479, 1.50899], cap → invertedSphericalCap[1.16088, 1.50899, rCap]|⟩
```

```
N[volume[tube]] → 6127.44
```

```
N[volumeFromDepth[tube, height[tube]]] → 6127.44
```

```
N[height[tube]] → 55.4
```

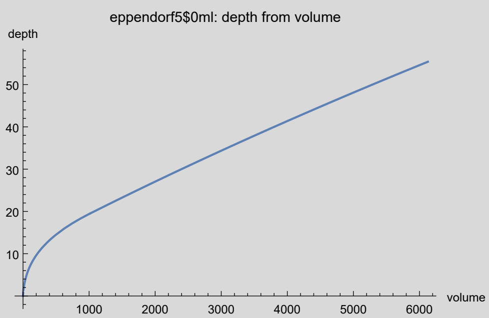
```
N[depthFromVolume[tube, volume[tube]]] → 55.4
```

```
N[2 radiusFromDepth[tube, height[tube]]] → 14.1726
```

N[simplify[depthFromVolume[tube, vol]]] →

$$\begin{cases} \frac{3.33494}{\left(-3.\,vol+\sqrt{116.524+9.\,vol^2}\right)^{1/3}} - 0.682784 \left(-3.\,vol + \sqrt{116.524 + 9.\,vol^2}\right)^{1/3} & vol \le 4.97137 \\ -4.52748 + 1.42939\,(39.9257 + 4.6465\,vol)^{1/3} & vol \le 1014.06 \\ -302.125 + 15.2946\,(8610.39 + 0.679419\,vol)^{1/3} & True \end{cases}$$

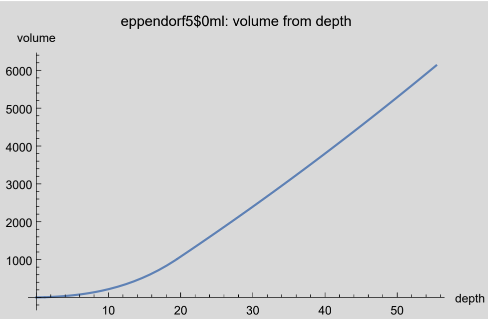
```
cFormat[simplify[depthFromVolume[tube, vol]]] →
 3.3349435128012708/cubeRoot(-3*vol + sqrt(116.52398253036392 +
   9*square(vol))) - 0.6827840632552957*cubeRoot(-3*vol + sqrt(116.52398253036392 + 9*square(vol)))   vol ≤ 4.97137
 -4.527482480392973 + 1.4293857242655184*cubeRoot(39.925707766396954 + 4.646502744123563*vol)        vol ≤ 1014.06
 -302.12525435323573 + 15.294554835805165*cubeRoot(8610.39131329194 + 0.6794188912396856*vol)        True
```



N[simplify[volumeFromDepth[tube, depth]]] →

$$\begin{cases} 3.57678\,depth + 0.523599\,depth^3 & depth \le 1.16088 \\ -1.75359 + depth\,(4.53169 + (1.00093 + 0.0736929\,depth)\,depth) & 1.16088 < depth \le 19.5033 \\ -1327.95 + depth\,(112.654 + (0.372872 + 0.000411388\,depth)\,depth) & True \end{cases}$$

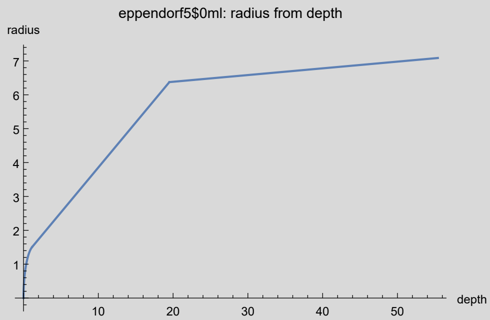
```
cFormat[simplify[volumeFromDepth[tube, depth]]] →
 3.5767759363317175*depth + 0.5235987755982988*cube(depth)                                      depth ≤ 1.16088
 -1.7535856283933002 + depth*(4.531691035316929 + (1.0009295574178767 + 0.07369287175618172*depth)*depth)      1.16088 < depth ≤ 19.5033
 -1327.9496657391219 + depth*(112.65414397006731 + (0.3728723181755454 + 0.00041138822701615246*depth)*depth)  True
```



N[simplify[radiusFromDepth[tube, depth]]] →

$$\begin{cases} 0.928123\sqrt{(3.6247 - 1.16088\,depth)\,depth} & depth \le 1.16088 \\ 1.20103 + 0.265276\,depth & depth \le 19.5033 \\ 5.98823 + 0.0198204\,depth & True \end{cases}$$

```
cFormat[simplify[radiusFromDepth[tube, depth]]] →
 0.9281234836336926*sqrt((3.624695781463986 - 1.1608830686450056*depth)*depth)  depth ≤ 1.16088
 1.2010337454342537 + 0.26527628779029744*depth                                 depth ≤ 19.5033
 5.988232439146341 + 0.01982036374935098*depth                                  True
```



idtTube

```
parts[tube] → ⟨|cylindrical → cylinder[38.3037, 4.16389], conical → invertedCone[3.69629, 4.16389], cap → cylinder[0, 0]|⟩
```

```
N[volume[tube]] → 2153.47
```

```
N[volumeFromDepth[tube, height[tube]]] → 2153.47
```

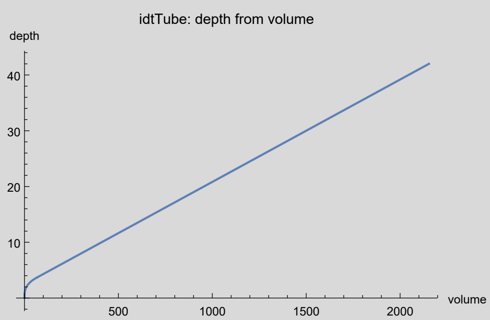
```
N[height[tube]] → 42.
```

```
N[depthFromVolume[tube, volume[tube]]] → 42.
```

```
N[2 radiusFromDepth[tube, height[tube]]] → 8.32778
```

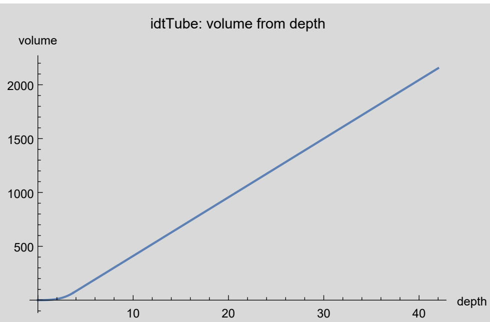
```
N[simplify[depthFromVolume[tube, vol]]] → { 0.                      vol ≤ 0.
                                          { 0.909568 vol^(1/3)      vol ≤ 67.1109
                                          { 2.46419 + 0.0183591 vol True
```

```
cFormat[simplify[depthFromVolume[tube, vol]]] → { 0                                                vol ≤ 0
                                                { 0.9095678851543723*cubeRoot(vol)                 vol ≤ 67.1109
                                                { 2.464193794602757 + 0.018359120058446303*vol     True
```



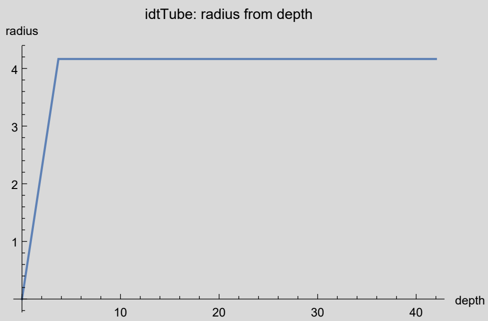

```
N[simplify[volumeFromDepth[tube, depth]]] → { 0.                        depth ≤ 0.
                                            { 1.32891 depth^3           depth ≤ 3.69629
                                            { -134.222 + 54.4688 depth  True
```

```
cFormat[simplify[volumeFromDepth[tube, depth]]] → { 0                                            depth ≤ 0
                                                  { 1.3289071745212766*cube(depth)               depth ≤ 3.69629
                                                  { -134.221781150621 + 54.46884147042437*depth  True
```




```
N[simplify[radiusFromDepth[tube, depth]]] → { 0.              depth ≤ 0.
                                            { 1.1265 depth    depth ≤ 3.69629
                                            { 4.16389         True
```

```
cFormat[simplify[radiusFromDepth[tube, depth]]] → { 0                           depth ≤ 0
                                                    1.126504715663486*depth     depth ≤ 3.69629
                                                    4.163888894893057           True
```



bioradPlateWell

```
parts[tube] → ⟨|cylindrical → cylinder[6.69498, 2.61859], conical → invertedFrustum[8.11502, 2.61859, 1.16608], cap → cylinder[0, 0]|⟩
```

```
N[volume[tube]] → 239.998
```

```
N[volumeFromDepth[tube, height[tube]]] → 239.998
```

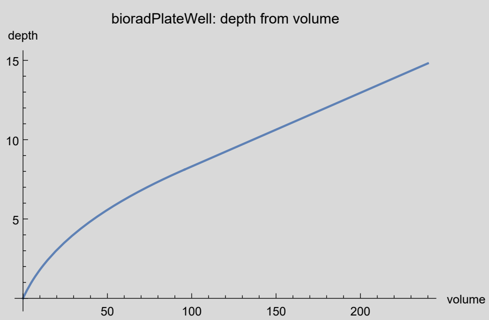
```
N[height[tube]] → 14.81
```

```
N[depthFromVolume[tube, volume[tube]]] → 14.81
```

```
N[2 radiusFromDepth[tube, height[tube]]] → 5.23718
```

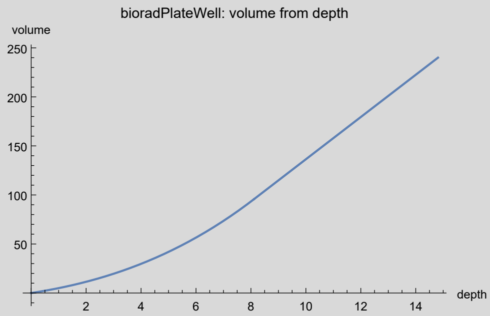
```
N[simplify[depthFromVolume[tube, vol]]] → { 0.                                              vol ≤ 0.
                                            -6.51474 + 2.78018 (12.8668 + 1.38705 vol)^(1/3)  vol ≤ 95.7748
                                            3.66906 + 0.046421 vol                          True
```

```
cFormat[simplify[depthFromVolume[tube, vol]]] →
 { 0                                                                                   vol ≤ 0
   -6.514739207958923 + 2.7801804906856553*cubeRoot(12.86684682940816 + 1.3870479041474308*vol)  vol ≤ 95.7748
   3.669055001226564 + 0.04642103427328387*vol                                         True
```



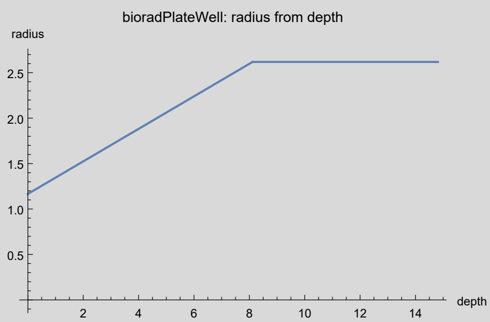

```
N[simplify[volumeFromDepth[tube, depth]]] → { 0.                                                   depth ≤ 0.
                                              depth (4.27174 + (0.655704 + 0.0335498 depth) depth)  depth ≤ 8.11502
                                              -79.0386 + 21.542 depth                              True
```

```
cFormat[simplify[volumeFromDepth[tube, depth]]] →
 { 0                                                                                  depth ≤ 0
   depth*(4.271740774393597 + (0.6557040332750236 + 0.033549771389874604*depth)*depth)  depth ≤ 8.11502
   -79.03863105734744 + 21.541958632651962*depth                                      True
```




```
N[simplify[radiusFromDepth[tube, depth]]] → { 0.                            depth ≤ 0.
                                              1.16608 + 0.178991 depth      depth ≤ 8.11502
                                              2.61859                       True
```

```
cFormat[simplify[radiusFromDepth[tube, depth]]] → { 0                                               depth ≤ 0
                                                    1.166077750282495 + 0.1789907029367993*depth   depth ≤ 8.11502
                                                    2.6185909188980574                              True
```



Pipettes

Similarly, we define copy-pasteable radial clearance functions.

```
(tips = {
      "opentrons_96_tiprack_10ul" → opentrons$10μl$tipM0,
      "opentrons_96_tiprack_300ul" → opentrons$300μl$tipM1
     } // Association) // Normal // ColumnForm
```

```
opentrons_96_tiprack_10ul → pipetteTip[invertedFrustum[39.2, 2.5, 0.75]]
opentrons_96_tiprack_300ul → pipetteTip[invertedFrustum[1.35, 3.455, 3.12], invertedFrustum[16.07, 3.12, 3.055], invertedFrustum[16.49, 2.47, 1.955
```

```
(pipettes = {
      "p50_single_v1.4" → p50M0
     } // Association) // Normal // ColumnForm
```

```
p50_single_v1.4 → pipette[invertedFrustum[60, 6.115, 5.67], invertedFrustum[3.05, 5.67, 4.705], cylinder[3.32, 2.555]]
```

```
tipUsage = {
  {"p50_single_v1.4", "opentrons_96_tiprack_300ul", {falcon15ml, falcon50ml, eppendorf1$5ml, eppendorf5$0ml, idtTube, bioradPlateWell}}
 }
```

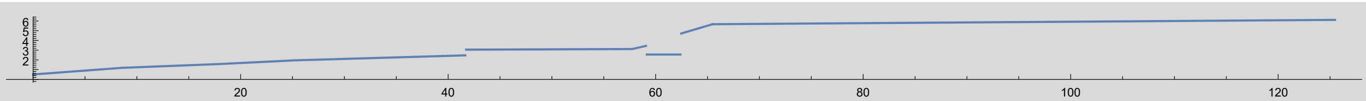
```
{{p50_single_v1.4, opentrons_96_tiprack_300ul, {falcon15ml, falcon50ml, eppendorf1$5ml, eppendorf5$0ml, idtTube, bioradPlateWell}}}
```

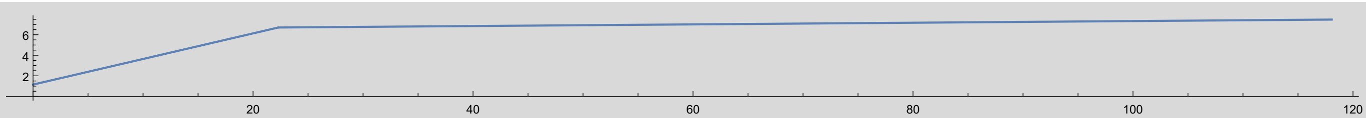
```
Clear[printAndPlot]
printAndPlot[pipetteModelName_, tipName_, tubeName_] := Block[{tube, simplify, tip, pip, mounted, minClearance, depth},
  tube = tubes[tubeName];
  simplify[fn_] := FullSimplify[fn, assumptions[tube]];

  CellPrint[TextCell[ToString[StringForm["``: ``: ``", pipetteModelName, tipName, tubeName]], "Subsubsection"]];

  pip = pipettes[pipetteModelName];
  tip = tips[tipName];
  mounted = mountTip[pip, tip];

  minClearance = minClearanceFromDepth[tube, mounted, depth];

  test @ N @ simplify @ minClearance;
  test @ cFormat @ simplify @ minClearance;

  cellPrint @ plotProfile[mounted];
  cellPrint @ plotProfile[tube];
  cellPrint @ Plot[minClearance, {depth, 0, height[tube]},
    PlotLabel → "min radial clearance vs depth", AxesOrigin → {0, 0}, AxesLabel → {"depth", "clearance"}];
 ]
printAndPlot[{pipetteModelName_, tipName_, tubeNames__List}] := printAndPlot[pipetteModelName, tipName, #] & /@ tubeNames
printAndPlot /@ tipUsage;
```

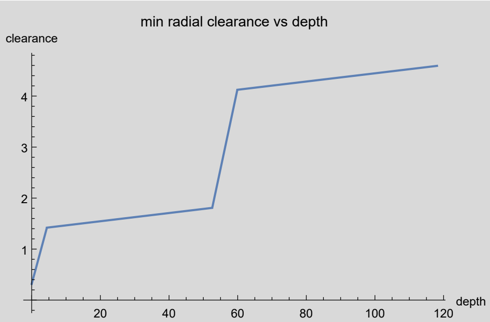
p50_single_v1.4: opentrons_96_tiprack_300ul: falcon15ml

```
N[simplify[minClearance]] →
  0.318101 + 0.249291 depth      0. < depth < 4.42012
  1.38438 + 0.00805941 depth     4.42012 ≤ depth < 52.59
  -14.8309 + 0.316393 depth      52.59 ≤ depth < 59.9064
  3.64027 + 0.00805941 depth     59.9064 ≤ depth ≤ 118.07
  ∞                              depth > 118.07
  0.                             True
```

```
cFormat[simplify[minClearance]] →
  0.3181014675267553 + 0.2492912278496944*depth      0 < depth < 4.42012
  1.3843756405067387 + 0.008059412406212692*depth    4.42012 ≤ depth < 52.59
  -14.830911008591517 + 0.3163934426229509*depth     52.59 ≤ depth < 59.9064
  3.640273194181542 + 0.008059412406212692*depth     59.9064 ≤ depth ≤ 118.07
  DirectedInfinity(1)                                depth > 118.07
  0                                                  True
```



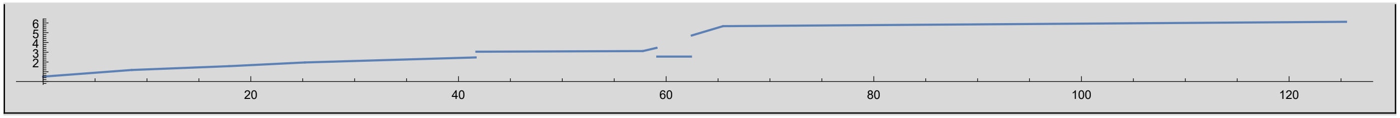


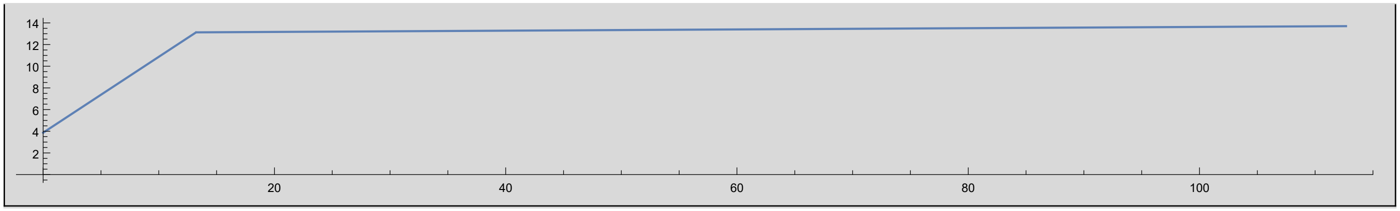


p50_single_v1.4: opentrons_96_tiprack_300ul: falcon50ml

```
N[simplify[minClearance]] →
  3.03677 + 0.700208 depth       0. < depth < 6.69969
  7.67825 + 0.00741667 depth     6.69969 ≤ depth < 47.19
  -6.90236 + 0.316393 depth      47.19 ≤ depth < 54.9388
  10.164 + 0.00574987 depth      54.9388 ≤ depth ≤ 112.67
  ∞                              depth > 112.67
  0.                             True
```

```
cFormat[simplify[minClearance]] →
  3.036768510671534 + 0.7002075382097096*depth      0 < depth < 6.69969
  7.678249659109892 + 0.007416666666666305*depth    6.69969 ≤ depth < 47.19
  -6.90236439826716 + 0.3163934426229509*depth      47.19 ≤ depth < 54.9388
  10.164032546012043 + 0.0057498667891316*depth     54.9388 ≤ depth ≤ 112.67
  DirectedInfinity(1)                               depth > 112.67
  0                                                 True
```

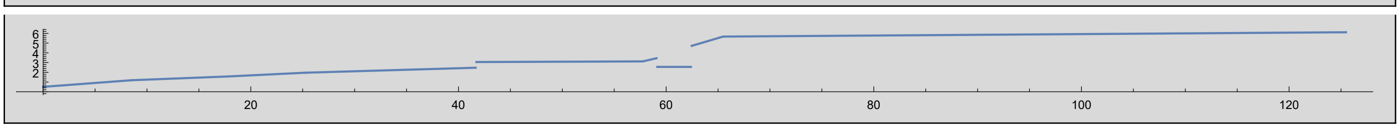


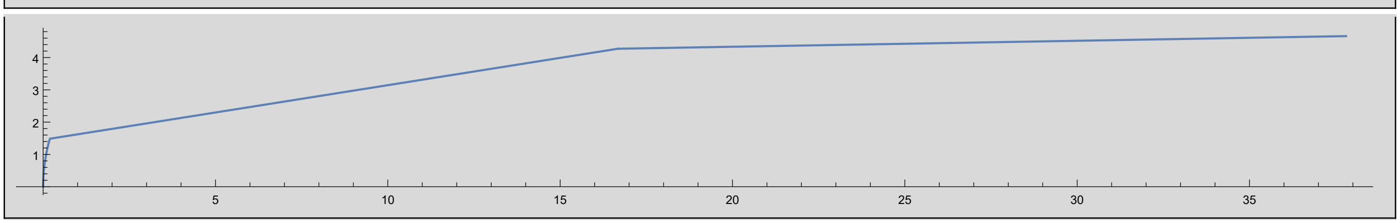


p50_single_v1.4: opentrons_96_tiprack_300ul: eppendorf1$5ml

```
N[simplify[minClearance]] →
  -0.505 + 2.26986 √((2.24622 - 0.194089 depth) depth)   0.022078 < depth < 0.114975
  0.623346 + 0.169045 depth                              0.114975 ≤ depth < 12.2688
  2.31416 + 0.031231 depth                               12.2688 ≤ depth < 12.6
  2.05178 + 0.0520548 depth                              12.6 ≤ depth < 19.9
  2.25939 + 0.0416222 depth                              19.9 ≤ depth ≤ 37.8
  ∞                                                      depth > 37.8
  0.                                                     True
```

```
cFormat[simplify[minClearance]] →
  -0.5049999999999997 + 2.2698647709367252*sqrt((2.2462186978285 - 0.19408860160231214*depth)*depth)   0.022078 < depth < 0.114975
  0.623346170657346 + 0.16904507285715673*depth                                                         0.114975 ≤ depth < 12.2688
  2.314155991679301 + 0.031231049120679123*depth                                                        12.2688 ≤ depth < 12.6
  2.0517767996409555 + 0.05205479452054796*depth                                                        12.6 ≤ depth < 19.9
  2.25938546033305 + 0.04162219850586984*depth                                                          19.9 ≤ depth ≤ 37.8
  DirectedInfinity(1)                                                                                   depth > 37.8
  0                                                                                                     True
```

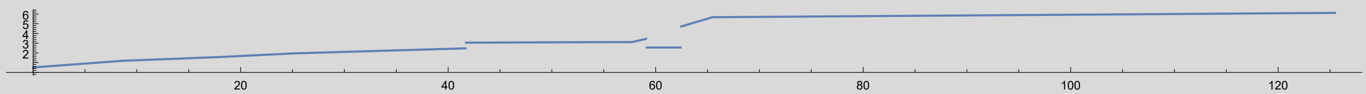


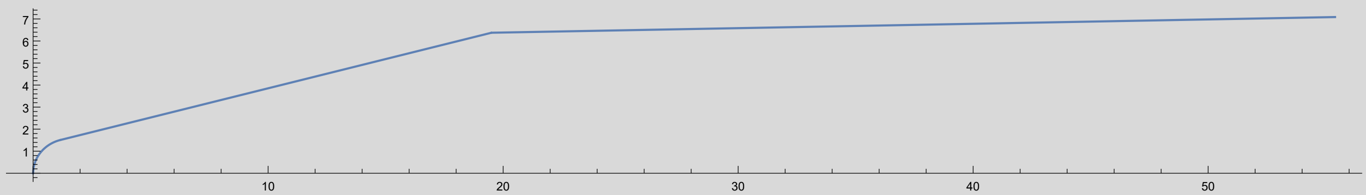


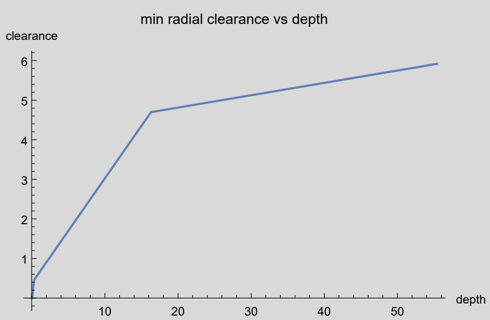
p50_single_v1.4: opentrons_96_tiprack_300ul: eppendorf5$0ml

```
N[simplify[minClearance]] →
  -0.505 + 0.928123 √((3.6247 - 1.16088 depth) depth)   0.0839332 < depth < 0.333631
  0.371071 + 0.265276 depth                              0.333631 ≤ depth ≤ 16.3089
  4.1881 + 0.031231 depth                                16.3089 < depth ≤ 55.4
  ∞                                                      depth > 55.4
  0.                                                     True
```

```
cFormat[simplify[minClearance]] →
  -0.5049999999999997 + 0.9281234836336926*sqrt((3.624695781463986 - 1.1608830686450056*depth)*depth)  0.0839332 < depth < 0.333631
  0.37107109868932375 + 0.26527628779029744*depth                                                      0.333631 ≤ depth ≤ 16.3089
  4.188102907415874 + 0.031231049120679123*depth                                                       16.3089 < depth ≤ 55.4
  DirectedInfinity(1)                                                                                  depth > 55.4
  0                                                                                                    True
```



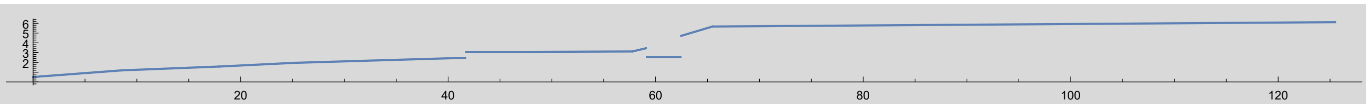


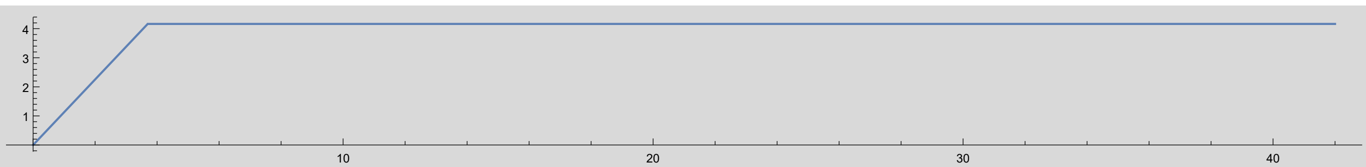


p50_single_v1.4: opentrons_96_tiprack_300ul: idtTube

```
N[simplify[minClearance]] →
  -0.505 + 1.1265 depth          0.448289 < depth ≤ 1.99878
  1.68421 + 0.031231 depth       1.99878 < depth ≤ 42.
  ∞                              depth > 42.
  0.                             True
```

```
cFormat[simplify[minClearance]] →
  -0.5049999999999997 + 1.126504715663486*depth     0.448289 < depth ≤ 1.99878
  1.6842072696656454 + 0.031231049120679123*depth   1.99878 < depth ≤ 42.
  DirectedInfinity(1)                               depth > 42.
  0                                                 True
```

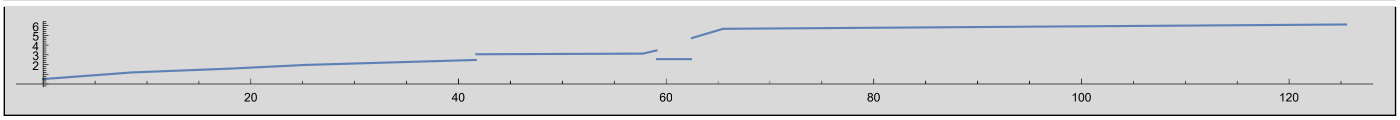


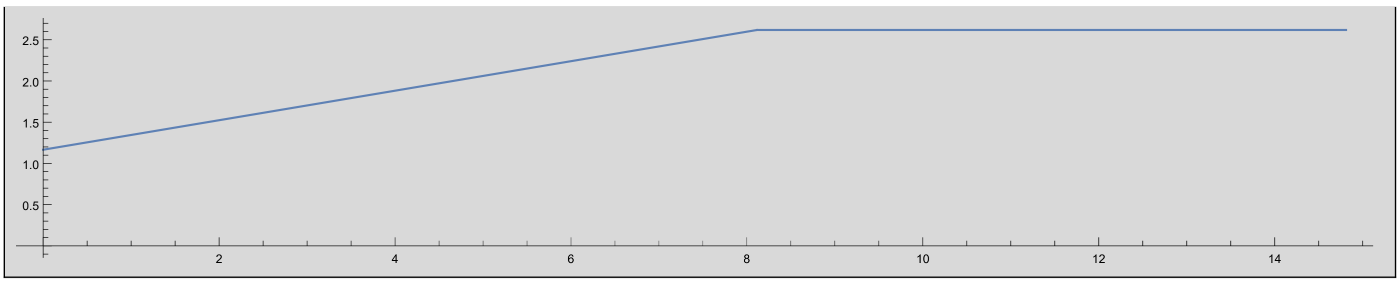


p50_single_v1.4: opentrons_96_tiprack_300ul: bioradPlateWell

```
N[simplify[minClearance]] → { 0.661078 + 0.178991 depth    0. < depth < 3.72084
                              1.1722 + 0.0416222 depth     3.72084 ≤ depth ≤ 6.28
                              0.932958 + 0.0797186 depth   6.28 < depth ≤ 14.81
                              ∞                            depth > 14.81
                              0.                           True
```

```
cFormat[simplify[minClearance]] → { 0.6610777502824954 + 0.1789907029367993*depth   0 < depth < 3.72084
                                    1.1722035122811951 + 0.04162219850586984*depth  3.72084 ≤ depth ≤ 6.28
                                    0.9329578591090766 + 0.07971864009378668*depth  6.28 < depth ≤ 14.81
                                    DirectedInfinity(1)                             depth > 14.81
                                    0                                               True
```





Comparing Models of Tubes

As a matter mostly of historical interest (at this point), we compare alternative models of several of our tubes.

Comparing 1.5 mL Eppendorf Tube Models

The fitted Eppendorf model clearly is better.

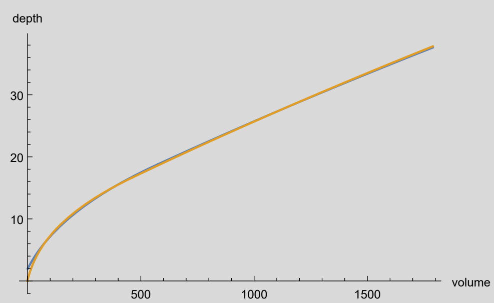
```
example2 = fittedEppendorf1$5M0;
example3 = fittedEppendorf1$5M1;
test @ example2;
test @ example3;
expr2 = depthFromVolume[example2, v]
expr3 = depthFromVolume[example3, v]
Row @ { Plot[{expr2, expr3}, {v, 0, volume[example3]}, AxesLabel → {"volume", "depth"}, ImageSize → Medium],
  Spacer[20],
  Plot[{expr2 - expr2, expr3 - expr2}, {v, 0, volume[example3]}, AxesLabel → {"volume", "Δdepth"}, ImageSize → Medium],
  Spacer[20],
   Show[ListPlot[{eppendorf15Data}, AxesLabel → {"vol", "depth"}, PlotRange → All, AxesOrigin → {0, 0}, ImageSize → Medium],
    Plot[{depthFromVolume[example2, v], depthFromVolume[example3, v]}, {v, 0, volume[example3]}]]}
```

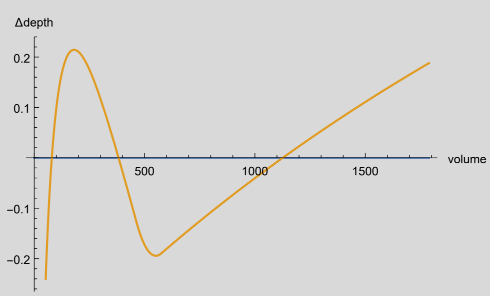
```
example2 →
 conicalTestTube[invertedFrustum[18.9894, 4.70751, 4.35636], invertedFrustum[16.8419, 4.35636, 2.1099], unknownShape[1.96866, 0.550217]]
```

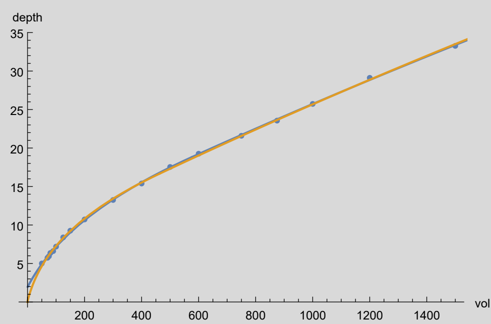
```
example3 → conicalTestTube[invertedFrustum[21.1258, 4.66267, 4.272],
  invertedFrustum[16.4801, 4.272, 1.48612], invertedSphericalCap[0.194089, 1.48612, rCap]]
```

$$\begin{cases} \begin{cases} 0 & v \le 0 \\ 1.96866 & v \ge 0.550217 \\ \text{Indeterminate} & \text{True} \end{cases} & v \le 0.550217 \\ -13.8495 + 2.9248\,(157.009 + 2.14521\,v)^{1/3} & v \le 575.88 \\ -216.767 + 20.2694\,(1376.83 + 0.33533\,v)^{1/3} & \text{True} \end{cases}$$

$$\begin{cases} \frac{3.23462}{\left(-3\,v + \sqrt{106.321 + 9\,v^2}\right)^{1/3}} - \frac{\left(-3\,v + \sqrt{106.321 + 9\,v^2}\right)^{1/3}}{\pi^{1/3}} & v \le 0.677156 \\ -8.59717 + 2.32458\,(52.2891 + 2.66032\,v)^{1/3} & v \le 463.316 \\ -214.342 + 19.5617\,(1474.21 + 0.373056\,v)^{1/3} & \text{True} \end{cases}$$





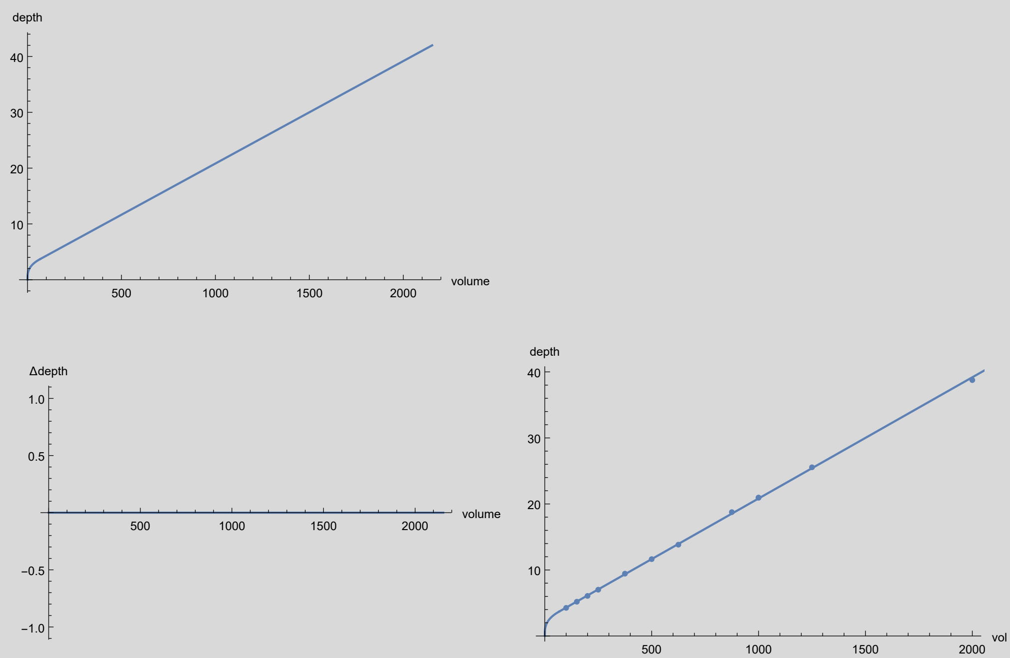


Comparing IDT Tube Models

```
example2 = tubes[idtTube];
test @ example2;
expr2 = depthFromVolume[example2, v]
Row @ {Plot[{expr2}, {v, 0, volume[example2]}, AxesLabel → {"volume", "depth"}, ImageSize → Medium],
  Spacer[20],
  Plot[{expr2 - expr2}, {v, 0, volume[example2]}, AxesLabel → {"volume", "Δdepth"}, ImageSize → Medium],
  Spacer[20],
   Show[ListPlot[{idtData}, AxesLabel → {"vol", "depth"}, PlotRange → All, AxesOrigin → {0, 0}, ImageSize → Medium],
    Plot[{depthFromVolume[example2, v]}, {v, 0, volume[example2]}]]}
```

```
example2 → conicalTestTube[cylinder[38.3037, 4.16389], invertedCone[3.69629, 4.16389], cylinder[0, 0]]
```

$$\begin{cases} 0 & v \le 0 \\ 0.909568\, v^{1/3} & v \le 67.1109 \\ 3.69629 - 0.0183591\,(67.1109 - v) & \text{True} \end{cases}$$



Comparing Bio-rad Plate models

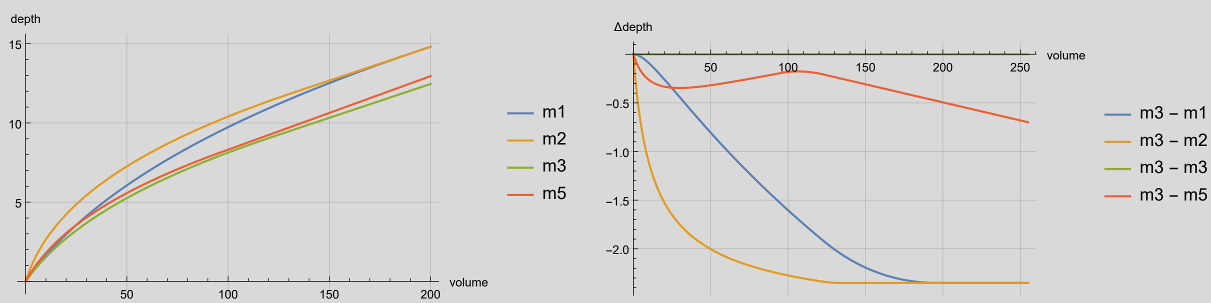
```
examplem1 = modelBioRad1[];
examplem2 = modelBioRad2[];
examplem3 = modelBioRad3[];
{ignored, examplem5} = modelBioRad5[];
test @ examplem1;
test @ examplem2;
test @ examplem3;
test @ examplem5;
exprm1 = depthFromVolume[examplem1, v];
exprm2 = depthFromVolume[examplem2, v];
exprm3 = depthFromVolume[examplem3, v];
exprm5 = depthFromVolume[examplem5, v];
Row @ { Plot[{exprm1, exprm2, exprm3, exprm5}, {v, 0, 200},
    AxesLabel → {"volume", "depth"}, PlotLegends → {"m1", "m2", "m3", "m5"}, GridLines → Automatic, ImageSize → Medium],
  Spacer[20],
  Plot[{exprm3 - exprm1, exprm3 - exprm2, exprm3 - exprm3, exprm3 - exprm5}, {v, 0, volume[examplem3]}, AxesLabel → {"volume", "Δdepth"},
    PlotLegends → {"m3 - m1", "m3 - m2", "m3 - m3", "m3 - m5"}, PlotRange → All, GridLines → Automatic, ImageSize → Medium]}
```

```
examplem1 → conicalTestTube[cylinder[0.150026, 2.73], invertedFrustum[14.66, 2.73, 1.32], cylinder[0, 0]]
```

```
examplem2 → conicalTestTube[cylinder[2.83192, 2.73], invertedFrustum[11.9781, 2.73, 0.886397], cylinder[0, 0]]
```

```
examplem3 → conicalTestTube[cylinder[5.64908, 2.73], invertedFrustum[9.16092, 2.73, 1.32], cylinder[0, 0]]
```

```
examplem5 → conicalTestTube[cylinder[6.69498, 2.61859], invertedFrustum[8.11502, 2.61859, 1.16608], cylinder[0, 0]]
```



Comparing 15mL Falcon Tube models

```
example2 = tubes[falcon15ml];
test @ example2;
expr2 = depthFromVolume[example2, v]
Row @ {Plot[{expr2}, {v, 0, volume[example2]}, AxesLabel → {"volume", "depth"}, ImageSize → Medium],
  Spacer[20],
  Plot[{expr2 - expr2}, {v, 0, volume[example2]}, AxesLabel → {"volume", "Δdepth"}, ImageSize → Medium],
  Spacer[20],
   Show[ListPlot[{falconData}, AxesLabel → {"vol", "depth"}, PlotRange → All, AxesOrigin → {0, 0}, ImageSize → Medium],
    Plot[{depthFromVolume[example2, v]}, {v, 0, volume[example2]}]]}
```

```
example2 → conicalTestTube[invertedFrustum[95.7737, 7.47822, 6.70634], invertedFrustum[22.2963, 6.70634, 1.14806], cylinder[0, 0]]
```

$$\begin{cases} 0 & v \le 0 \\ -4.60531 + 1.42522\,(33.739 + 5.30776\,v)^{1/3} & v \le 1260.65 \\ -809.817 + 27.1195\,(27957.8 + 0.737091\,v)^{1/3} & \text{True} \end{cases}$$