Drawing Surfaces of Revolution with Ti*k*Z

A Tutorial for `surfacesOfRevolution(AndH).sty`

Daniel Naie

April 3, 2026

Contents

1 Introduction

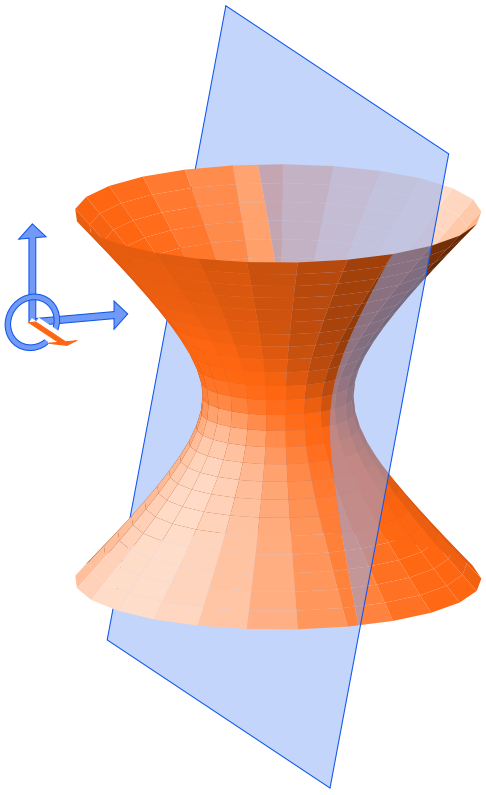

Figure 1: The hyperboloid intersection with a plane, $x + by + c = 0$. The point of view is modified globally. In case a rotation is performed on the hyperboloid, it must also be performed on the plane's equation. (For the code, see subsection 8.4.)

This tutorial explains how to use three companion style files to draw surfaces of revolution with Ti*k*Z:

- `surfacesAndCurves.sty` — defines the built-in generating curves (sphere, cone, torus, ...) and the parameters that control their shape.
- `surfacesOfRevolution.sty` — the main engine: sets up the observer and light source, discretises the surface into a grid, computes visibility and shading, and provides the drawing commands.
- `surfacesOfRevolutionAndH.sty` — an extension that clips a surface to a half-space defined by a linear inequality $H > 0$, making it possible to show cross-sections or to combine two surfaces that intersect.

All three files must be placed where LaTeX can find them (e.g. in the same directory as your document), and loaded in the preamble:

```
\usepackage{tikz}
\usepackage{tikz-3dplot}
\usetikzlibrary{math, calc}

\usepackage{surfacesAndCurves}
\usepackage{surfacesOfRevolution}
\usepackage{surfacesOfRevolutionAndH}  % if hyperplane clipping is needed
```

The drawings can be slow for large numbers of meridians and parallels. Using Ti*k*Z's externalisation library is strongly recommended:

```
\usetikzlibrary{external}
\tikzexternalize[prefix=z_ext/]
```

Then surround each picture with `\tikzexternalenable` / `\tikzexternaldisable` to cache it as a separate PDF.

2 Coordinate system and conventions

We introduce the world frame, the y-axis as revolution axis, the `view` style (longitude/latitude), the layer system with `\setlayers`, and a pic object representing the world frame as seen by the observer.

2.1 The world frame

The package works in a right-handed 3D coordinate system (x, y, z). In the default position, the y-axis is vertical and the z-axis points outside the screen.

The *default* revolution axis will be the y-axis. The generating curve (the profile) lives in the yz-plane, parameterised by a variable t ranging from `t_ini` to `t_end`:

$$\gamma(t) = (0,\ y(t),\ z(t)).$$

Rotating γ around the y-axis by angle $\theta \in [0°, 360°]$ gives the surface

$$(x, y, z) = (z(t)\sin\theta,\ y(t),\ z(t)\cos\theta).$$

2.2 The observer

The observer's direction is set with the `view` style, which takes two arguments: *longitude* λ and *latitude* ϕ (both in degrees). Longitude is measured in the xz-plane from the z-axis; latitude is the angle above the horizontal plane.

```
\begin{tikzpicture}[view={28}{20}]   % longitude 28°, latitude 20°
  ...
\end{tikzpicture}
```

Internally, `view` sets Ti*k*Z's canvas vectors `x`, `y`, `z` so that coordinates expressed in the world frame are projected correctly onto the page. It also stores the observer's unit direction vector (`\toxx`, `\toyy`, `\tozz`), used later to determine which faces are visible.

2.3 Layers

The package relies on `pgfonlayer` for correct depth ordering. You must declare the layers at the beginning of each picture using the convenience command `\setlayers`:

```
\setlayers{-2, -1, main, 1, 2}
```

This declares *and* activates the layers in back-to-front order. The main drawing canvas is always called `main`. You are free to use as many numeric layers as you need; typical choices are `-1, main, 1` (three layers) or `-2, -1, main, 1, 2` (five layers). When drawing a surface you will tell the package which layer to use for the visible ("above") and the hidden ("below") faces.

Remark. The layer `main` cannot be used in the `rSurface` pic type objects. I do not know how to handle `main` in the code.

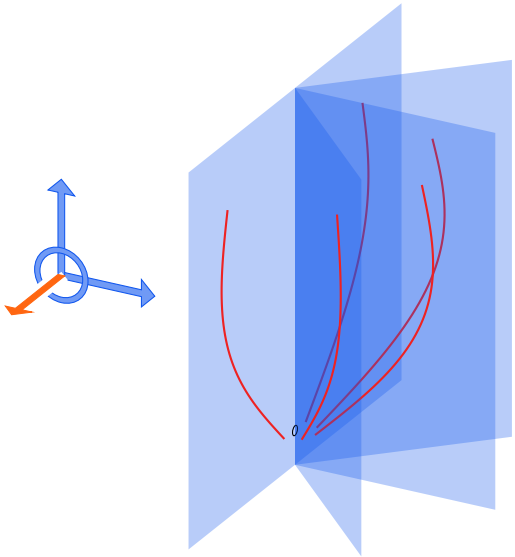

Figure 2: The effect of `\showCoordinates` and the curve defined in subsection 3.3. The curve is drawn in the half plane defined by $x = 0$ and $z \geq 0$. This half plane is then rotated around the y-axis four times.

2.4 The coordinate axes indicator: showCoordinates

The showCoordinates pic draws a small set of coordinate axes, useful for debugging or for pedagogical figures. It is placed at a point in the picture and takes three arguments:

```
\path (-3.5, 2.5, 0) pic {showCoordinates={1.2}{-1}{-2}};
%  arguments:  length   fg-layer  bg-layer
```

The x- and y-axes appear in blue with a small annulus at the origin; the z-axis in orange. The two layer arguments control depth: the axis that is "in front" goes on the foreground layer.

3 Generating curves

In this section: the anatomy of a yFoo/zFoo function pair; all six built-in curves (sphere, cone, hyperboloids, ellipsoid, torus) with their parameter ranges and shape keys; how to write a custom curve.

3.1 Anatomy of a generating curve

A generating curve is defined by two \tikzmath functions:

- yName(\tvar) — the y-coordinate of the profile as a function of the parameter t,
- zName(\tvar) — the z-coordinate (the *radius* of revolution at height y).

The name Name is a free identifier that will be passed to the sor style (see Section 4).

3.2 Built-in curves in surfacesAndCurves.sty

The file surfacesAndCurves.sty provides six ready-made profile curves for well-known surfaces.

Sphere

Parameter $t \in [-90°, 90°]$ (latitude). The radius is controlled by the key sphereRadius (default 1).

```
function ySphere(\tvar) { return \sphRadius * sin(\tvar); };
function zSphere(\tvar) { return \sphRadius * cos(\tvar); };
```

Set the radius with [sphereRadius=3] inside a scope.

Cone (truncated)

Parameter $t \in [0, 1]$. Keys: baseRadius, topRadius, height.

```
function yCone(\tvar) { return \tvar * \height; };
function zCone(\tvar) { return \baseRadius + \tvar*(\topRadius-\baseRadius); };
```

Hyperboloid of one sheet

Parameter $t \in [-y_{\max}, y_{\max}]$. Keys: `z half axis` ($a$), `y half axis` ($b$).

$$z = a\sqrt{1 + (y/b)^2}.$$

Hyperboloid of two sheets

Parameter $t \in [0, z_{\max}]$. The surface is the upper nappe.

Ellipsoid

Parameter $t \in [-90°, 90°]$ (latitude). Keys: `y half axis` (b), `z half axis` (a).

$$y = b\sin t, \quad z = a\cos t.$$

Torus

Parameter $t \in [0°, 360°]$ (tube angle). Keys: `tMajorRadius` (R), `tMinorRadius` (r).

$$y = r\sin t, \quad z = R + r\cos t.$$

3.3 Defining a custom curve

You can define your own curve anywhere before the picture. For example, a "vase" profile (see Fig. 2):

```
\tikzmath{
  function yVase(\tvar) { return \tvar; };          % t in [0, 4]
  function zVase(\tvar) {
    return 0.3 + 0.8*sin(45*\tvar) + 0.4*\tvar;
  };
}
```

The name `Vase` is then passed to `sor` as `name=Vase`. The only constraint is that $z(t) \geq 0$ everywhere (it is the radius of revolution), and that both functions are defined on [`t_ini`, `t_end`].

4 The `sor` style and basic drawing

In this section: every argument of `sor` explained in detail; the `rSurface` pic with its `above/below/actions` arguments; drawing meridians and parallels individually or in families; `partial rSurface` for sectors.

4.1 The `sor` style

The `sor` style is the heart of the package. It is applied as a Ti*k*Z option, typically inside a `scope`:

```
\begin{scope}[sphereRadius=2,
  sor={name=Sphere,
       meridians=24, parallels=18,
       t_ini=-90, t_end=90,
       rAxis={0, 0, 0}}]
  ...  pic objects go here ...
\end{scope}
```

Its arguments are:

name=*Name* The identifier shared by the functions `yName` and `zName`.

meridians=*M* The number of meridians. The surface is divided into M sectors; meridian 0 coincides with meridian M. More meridians give a smoother surface but slower compilation. Typical values: 12–36.

parallels=*P* The number of *intervals* between consecutive parallels; there are $P + 1$ parallel circles. Typical values: 8–28.

t_ini=*value*, t_end=*value* The parameter range. The $P + 1$ parallels are uniformly spaced from `t_ini` to `t_end`.

rAxis={*ax, az, ay*} Three Euler angles (in degrees) that tilt the revolution axis away from y. Specifically, the axis is obtained by applying $R_x(a_x)$, then $R_z(a_z)$, then $R_y(a_y)$ to the y-axis. The default `{0, 0, 0}` keeps the y-axis vertical. See Section 6 for examples.

Internally, `sor` computes the world coordinates of all grid points $P_{k,j}$ (meridian index $k = 0, \ldots, M$; parallel index $j = 0, \ldots, P$), the outward unit normal vector of each quadrilateral face, and the inner products of that normal with the observer and sun directions. These values are stored in arrays accessible to the pic objects described below.

4.2 The `rSurface` pic: wire-frame

The simplest drawing mode fills each visible face with a solid colour and optionally draws its boundary. No light source is needed. The output of the code below is the first sphere in Fig. 3.

```
\begin{tikzpicture}[view={28}{20}]
  \setlayers{-1, main, 1}

  \begin{scope}[sphereRadius=2,
    sor={name=Sphere, meridians=18, parallels=14,
         t_ini=-90, t_end=90, rAxis={0, 0, 0}}]
    \draw (0,0,0) pic[B]
    {rSurface={above=1, below=-1,
               actions={draw, fill=W, fill opacity=.6}}};
  \end{scope}
\end{tikzpicture}
```

above=*layer* The layer number used for faces whose outward normal points *towards* the observer (visible faces).

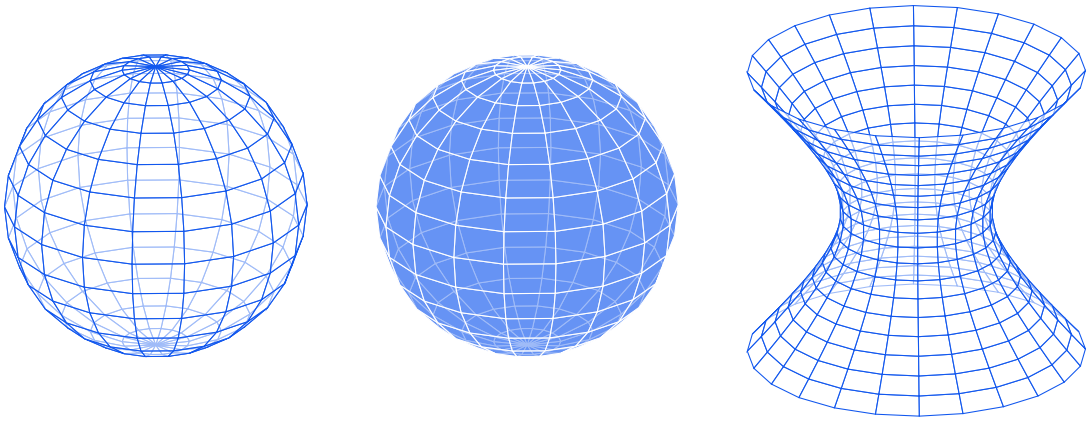

Figure 3: For the second sphere, `color=W`, `fill=B`, and `fill opacity=.4`.

`below=`*layer* The layer for hidden faces. Using a negative number places them behind `main`.

`actions={...}` Standard Ti*k*Z path actions applied to every quadrilateral face: `draw`, `fill`, `fill opacity`, etc. The current colour (set by `pic[COLOR]`) is available as "`.`" inside `actions`.

The colour passed to `pic[COLOR]` controls the *stroke* of the edges (and also fills, if you use `fill=.` in `actions`). Hidden faces receive the same treatment but land on a lower layer, so they appear behind all visible ones.

Example 4.1. A hyperboloid of one sheet, wire-frame with semi-transparent fill (see Fig. 3).

```
\begin{tikzpicture}[view={28}{15}]
  \setlayers{-1, main, 1}

  \begin{scope}[z half axis=1, y half axis=1,
    sor={name=Hyperboloid1, meridians=24, parallels=19,
      t_ini=-3, t_end=3, rAxis={0, 0, 0}}]
    \draw (0,0,0) pic[B]
    {rSurface={above=1, below=-1,
        actions={draw, fill=W, fill opacity=.6}}};
  \end{scope}
\end{tikzpicture}
```

4.3 Drawing meridians and parallels

Four pic objects let you add grid lines selectively.

`meridian={index=`k`, layer=`l`}` Draws a single meridian with index $k \in \{0, \dots, M-1\}$ on layer l.

`parallel={index=`j`, layer=`l`}` Draws a single parallel with index $j \in \{0, \dots, P\}$.

`meridians={initial=`k_0`, multiple=`s`, layer=`l`}` Draws meridians with indices $k_0, k_0 + s, k_0 + 2s, \dots$

`parallels={initial=`j_0`, multiple=`s`, layer=`l`}` Draws parallels with indices $j_0, j_0 + s, j_0 + 2s, \dots$

The colour is again given via `pic[COLOR]`. Meridians and parallels may be drawn *after* the surface so they appear on top, but the argument `level` allows a lot of flexibility. Note that these elements are thought of as reference elements in the drawing; there is no seen/unseen development for them.

```
\draw (0,0,0) pic[B]
    {rSurface={above=1, below=-1, actions={draw, fill=W, fill opacity=.5}}};
\draw (0,0,0) pic[R] {meridians={initial=0, multiple=6, layer=1}};
\draw (0,0,0) pic[O] {parallels={initial=0, multiple=3, layer=1}};
```

4.4 Partial surfaces: `partial rSurface`

To draw only a sector of a surface (a range of meridians), use `partial rSurface`:

```
  \draw (0,0,0) pic[B] {partial rSurface={above=1, below=-1,
      actions={draw, fill=B!20, fill opacity=.7},
      i from=3, i to=15}};
```

The arguments `i from` and `i to` are meridian indices. The sector drawn comprises the faces between those two meridians. This is useful for showing a cross-section or for combining two differently coloured halves of the same surface.

Example 4.2. A sphere with an opaque sector.

```
\begin{tikzpicture}[view={3}{31}]
  \setlayers{-2, -1, main, 1}

  \begin{scope}[sphereRadius=3,
    sor={name=Sphere, meridians=18, parallels=10,
      t_ini=-75, t_end=60, rAxis={0, 0, 0}}]
    %% Full sphere, light fill
    \path (0,0,0) pic[G]
    {rSurface={above=1, below=-2, actions={draw, fill=W, fill opacity=.4}}};
    %% A highlighted sector
    \path (0,0,0) pic[W]
    {partial rSurface={above=1, below=-1,
        actions={draw, fill=O!70, fill opacity=.95}, i from=3, i to=9}};
    %% special meridian and parallel
    \path (0,0,0) pic[DarkB] {meridian={index=3, layer=1}};
    \path (0,0,0) pic[DarkB] {parallel={index=10, layer=1}};
  \end{scope}
\end{tikzpicture}
```

5 Lighting: `rSurface L`

In this section: the `sun` style; the colour model (K/W blending, contrast, `local color threshold`, `max color cst`); a torus example with empty actions for a fully smooth surface.

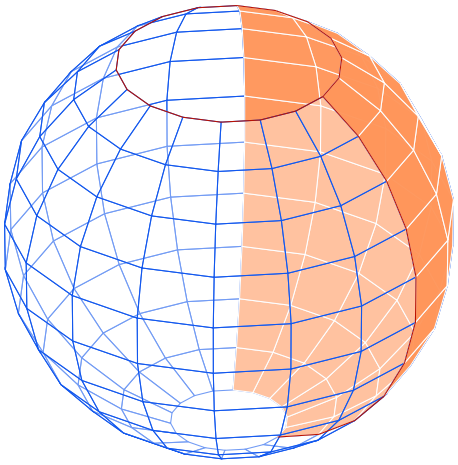

Figure 4: A sphere with an opaque sector. A meridian and a parallel are also shown in a different colour.

5.1 The `sun` style

Adding a light source enables smooth shading. The sun's direction is set with the `sun` style, which also takes longitude λ and latitude ϕ, plus a *contrast* value $c \in [0, 1]$:

```
\begin{tikzpicture}[view={28}{20}, sun={-50}{35}{.8}]
  \ldots
\end{tikzpicture}
```

If `sun` is not used, `contrast` defaults to -1, which tells the engine that no light source is active; in that case the `L` variants below behave identically to their unlit counterparts.

5.2 The lit pic: `rSurface L`

The `rSurface L` pic draws each face with a colour that depends on the angle between the face normal and the sun direction.

```
\begin{tikzpicture}[view={11}{25}, sun={-50}{35}{.7}]
  \setlayers{-1, main, 1}

  \begin{scope}[sphereRadius=2,
    sor={name=Sphere, meridians=36, parallels=28,
         t_ini=-90, t_end=90, rAxis={0, 0, 0}}]
    \draw (0,0,0) pic[B] {rSurface L={above=1, below=-1, actions={draw}}};
  \end{scope}
\end{tikzpicture}
```

The colour model works as follows. Two reference colours are set implicitly:

- `K` (black by default) — the dark-side colour.
- `W` (white by default) — the highlight colour.

For a face whose outward normal makes an angle with the sun direction parameterised by the inner product $\sigma \in [-1, 1]$:

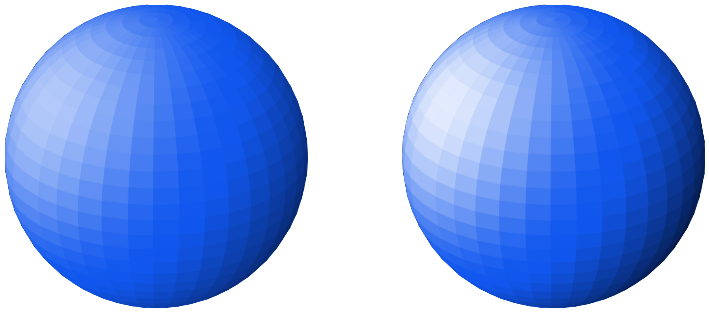

Figure 5: Spheres in full lightcoming fron the left. For the first sphere, the third argument of `sun` is 0.7 and for the second it is 0.9.

- If the face is in full sunlight ($\sigma \geq$ `local color threshold`, default 0.2), the face colour is blended towards `W` by an amount up to `max color cst` $\times c$ (a quadratic ramp).
- Otherwise it is blended towards `K` (linear ramp).

The *local* colour — the baseline at zero blending — is determined by the current colour set by `pic[COLOR]` and stored as `.` (the Ti*k*Z "current colour").

In practice: choose a medium-saturation colour for `pic[COLOR]`, keep `contrast` around 0.7–0.9, and the shading will look natural.

5.3 Adjustable shading parameters

These are `pgfkeys` set inside the `tikzpicture` options or a `scope`:

`local color threshold=`*val* Inner-product value at which the colour equals the local (unshaded) colour. Default: 0.2.

`max color cst=`*val* Maximum blending fraction towards `W` on the highlight side. Default: 0.99.

`contrast=`*val* Global intensity of the shading effect. Controlled via the third argument of `sun`.

Example 5.1. A torus with lighting.

```
\begin{tikzpicture}[view={-30}{25}, sun={-60}{40}{.85}]
  \setlayers{-1, main, 1}

  \begin{scope}[tMajorRadius=3, tMinorRadius=1,
    sor={name=Torus, meridians=48, parallels=24,
         t_ini=0, t_end=360, rAxis={0, 0, 0}}]
    \path (0,0,0) pic[R] {rSurface L={above=1, below=-1, actions={}}};
  \end{scope}
\end{tikzpicture}
```

Note that `actions={}` (empty) suppresses edge drawing, giving a fully smooth shaded surface. Adding `draw` in `actions` outlines each quadrilateral face.

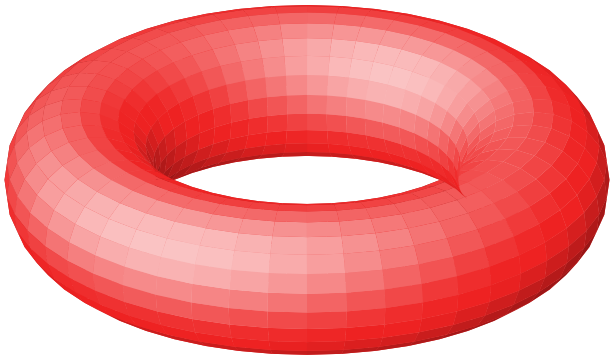

Figure 6: A torus with `actions={}`. The output is not correct at the inner right turning point. We shall come back to this issue in subsection 5.4.

5.4 Order of drawing the quadrilaterals and non convex surfaces

The current two-layer dispatch cannot handle the problem raised by the non-convex surfaces. We show here how to fix the problem for such an opaque surface: we just need to control the drawing order *within* the 'above' layer; see the first `partial rSurface L` in Fig. 7.

Of course, there are at least three limitations with this ad-hoc solution:

1. Some testing is necessary to decide which meridians to use for the reordering process.
2. The solution is not stable if either the number of meridians is changed, or the observer's 'position',
3. The solution does not work is transparancy is involved or if the surface is open by a cutting process (to be done—see Section **??** below).

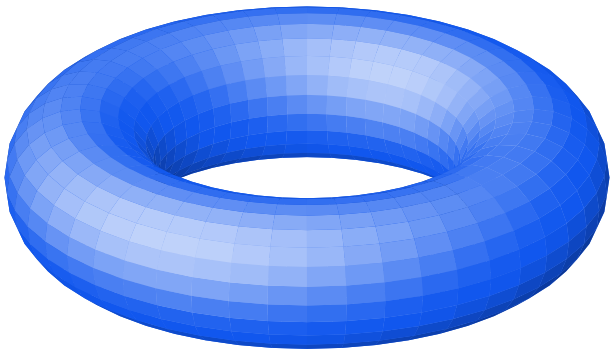

Figure 7: The torus again with reversed order of drawing the quadrilaterals on the right side.

```
\begin{tikzpicture}[view={-30}{25}, sun={-60}{40}{.75}]
  \setlayers{-1, main, 1}
  \begin{scope}[tMajorRadius=3, tMinorRadius=1,
    sor={name=Torus, meridians=48, parallels=24,
      t_ini=0, t_end=360, rAxis={0, 0, 0}}]
    \path (0,0,0) pic[0!20!R] {partial rSurface L={above=1, below=-1,
        actions={}, i from=23, i to=0}};
    \path (0,0,0) pic[0!20!R] {partial rSurface L={above=1, below=-1,
        actions={opacity=.5}, i from=23, i to=48}};
  \end{scope}
\end{tikzpicture}
```

6 Tilting the revolution axis: rAxis

The optional rAxis argument to sor specifies three Euler angles $\{a_x, a_y, a_z\}$ (in degrees) that tilt the revolution axis. The default axis is y; the tilted axis is obtained by

$$\text{tiltaxis} = R_y(a_y) \circ R_z(a_z) \circ R_x(a_x) \cdot \text{axes}\,.$$

A common use case is to rotate a surface so that its axis points in a direction other than vertical. For example, to tilt the axis 20° around z, one may use

```
sor={name=Hyperboloid2, ..., rAxis={0, 0, 20}}
```

The surface's grid points are recomputed in the tilted frame, but lighting and visibility are still computed in the original world frame, so the sun illuminates the scene correctly regardless of the tilt.

Example 6.1. Two hyperboloids of type 2 with different axis tilts, drawn side by side.

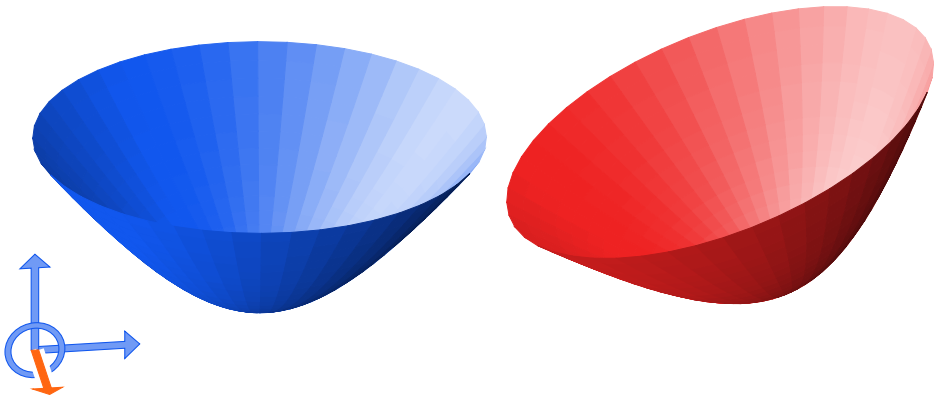

Figure 8: The use of the argument rAxis

```
\begin{tikzpicture}[view={-18}{25}, sun={-75}{35}{.8}]
  \setlayers{-1, main, 1}
  \path (-6, 2.5, 0) pic {showCoordinates={1.2}{-1}{-2}};
  %% Left: vertical axis
  \begin{scope}[y half axis=3, z half axis=2,
    sor={name=Hyperboloid2, meridians=24, parallels=12,
         t_ini=0, t_end=3, rAxis={0, 0, 0}}]
    \draw (-3,0,0) pic[B]
    {rSurface L={above=1, below=-1, actions={draw}}};
  \end{scope}

  %% Right: axis tilted 20^\circ around x
  \begin{scope}[y half axis=3, z half axis=2,
    sor={name=Hyperboloid2, meridians=24, parallels=12,
         t_ini=0, t_end=3, rAxis={0, 0, 20}}]
    \draw (5,0,0) pic[R]
    {rSurface L={above=1, below=-1, actions={draw}}};
```

```
  \end{scope}
\end{tikzpicture}
```

7 Combining multiple surfaces

Because `sor` stores its grid in global variables, you *cannot* call two `sor` scopes simultaneously. However, you can draw them sequentially inside the same `tikzpicture`, as long as each `scope` with `sor` is completely finished before the next begins. The layers handle correct depth ordering automatically.

Example 7.1. A cone resting on top of a cylinder (a truncated cone with equal radii).

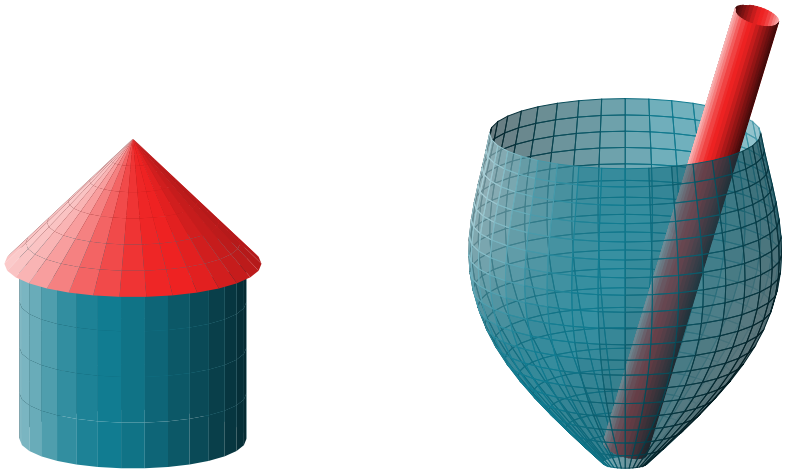

Figure 9: On the left, a cone and a cylinder—two scopes have been used. On the right the vase based on the curve introduced in Figure 2 and a cylinder. In the right draawing, the layers used are -2 and 2 for the vase and 1 and -1 for the cylinder.

```
\begin{tikzpicture}[view={30}{15}, sun={-55}{35}{.8}]
  \setlayers{-1, main, 1}

  %% Cylinder (cone with equal radii)
  \begin{scope}[baseRadius=1.5, topRadius=1.5, height=2.5,
    sor={name=Cone, meridians=30, parallels=4,
         t_ini=0, t_end=1, rAxis={0, 0, 0}}]
    \path (0,0,0) pic[B!50!G]
    {rSurface L={above=1, below=-1, actions={}}};
  \end{scope}

  %% Cone on top (base at y=2.5)
  \begin{scope}[baseRadius=1.7, topRadius=0, height=1.7,
    sor={name=Cone, meridians=30, parallels=6,
         t_ini=0, t_end=1, rAxis={0, 0, 0}}]
    \path (0,2.4,0) pic[R]
    {rSurface L={above=1, below=-1, actions={}}};
  \end{scope}
\end{tikzpicture}
```

Note that the second `pic` is placed at `(0, 2.5, 0)` to shift the cone upward. Ti*k*Z applies this shift before projecting to the canvas, so the 3-D positioning is exact.

8 Hyperplane clipping with `surfacesOfRevolutionAndH.sty`

8.1 Motivation

The standard `rSurface` always draws the *complete* surface. When two surfaces interpenetrate, or when you want to show a cross-section, you need to restrict the drawing to one side of a plane. The file `surfacesOfRevolutionAndH.sty` provides this functionality.

8.2 The `sor and H` style

`sor and H` is a drop-in replacement for `sor` with one extra mandatory argument, `cHyperplane`:

```
sor and H={name=Sphere,
  meridians=36, parallels=24,
  t_ini=10, t_end=90,
  rAxis={0, 0, 0},
  cHyperplane={a, b, c, d}}
```

The clipping half-space is $\{(x, y, z) : ax + by + cz + d \geq 0\}$. Only the part of the surface with $ax + by + cz + d \geq 0$ is drawn.

The algorithm works at the quadrilateral level. For each face it classifies all four vertices as *positive* $(+1)$, *zero* (0), or *negative* (-4) with respect to the hyperplane. The "drawing weight" of a face is the sum of its four vertex weights. Faces with weight ≥ 0 (all vertices non-negative) are drawn in full. Faces that straddle the boundary are clipped: new boundary points are computed by linear interpolation along each edge that crosses the hyperplane, and the resulting polygon is drawn instead of the original quadrilateral. Faces entirely in the negative half-space are omitted.

8.3 Pic objects

Four pic objects are available, parallel to those in `surfacesOfRevolution.sty`:

`rSurface and H` Unlit clipped surface (wire-frame or transparent fill).

`rSurface and HL` Lit clipped surface (uses the `sun` shading model).

`partial rSurface and H` Unlit, restricted to a meridian range.

`partial rSurface and HL` Lit, restricted to a meridian range.

Their arguments (`above`, `below`, `actions`, optionally `i from`/`i to`) are identical to their unclipped counterparts.

8.4 A first example and the code of the drawing from the introduction

New grid points are introduced when clipping a surface of revolution. The procedure is suggested in the figure below.

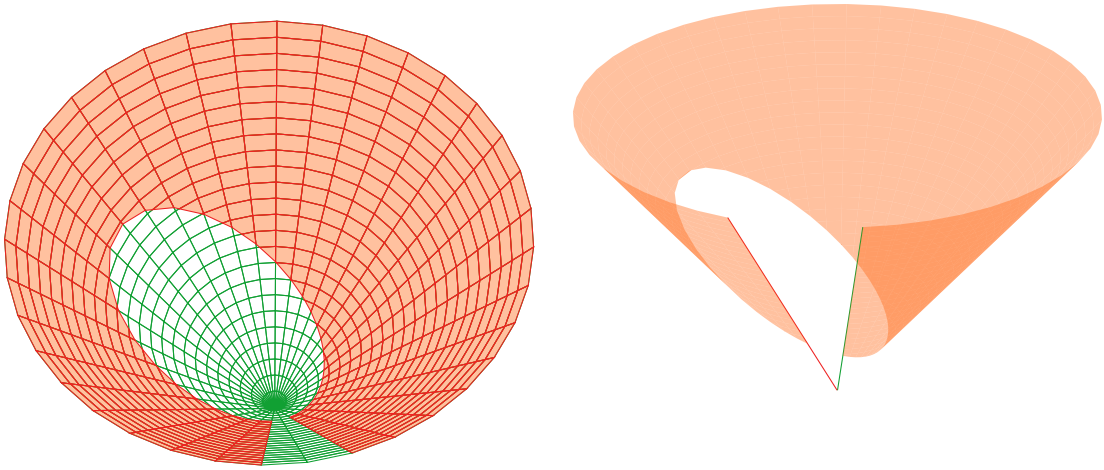

Figure 10: A closer look at the grid in the case of the clipping of a cone with 36 meridians and 24 parallels; the clipped cone is drawn in red and the unclipped, beneath, in green. On the right, the same surface with 48 meridians and 32 parallels with the grid hidden.

The code for the left drawing in Fig. 10:

```
\begin{tikzpicture}[view={2}{25}]
  \setlayers{-2, -1, main, 1, 2}

  % full cone
  \begin{scope}[height=4, baseRadius=.01, topRadius=3.5,
    sor={name=Cone, meridians=36, parallels=24, t_ini=0, t_end=1, rAxis={32, 0, 0}}]
    \path (0, 0, 0) pic[G] {rSurface={above=1, below=-2, actions={draw}}};
  \end{scope}

  % clipped cone
  \begin{scope}[height=4, baseRadius=.01, topRadius=3.5,
    sor and H={name=Cone, meridians=36, parallels=24,
      t_ini=0, t_end=1, rAxis={32, 0, 0}, cHyperplane={1, 1.7, 0.5, -1.9}}]
    \path (0, 0, 0) pic[R] {partial rSurface and H={above=1, below=-1,
        actions={draw, fill=0, fill opacity=.4}, i from=2, i to=36}};
  \end{scope}
\end{tikzpicture}
```

The code for the drawing in Fig. 1:

```
\begin{tikzpicture}[view={-20}{14}, sun={-70}{45}{.8},
  line cap=round, evaluate={%
    real \posH, \yMax, \xH, \yH, \zH;
    \cHt = .41;
    \cHy = .1;
    \yMax = 2.5;   % hyperplane's "dimensions"
    \yH = 1.7 *\yMax;
    \zH = 1.6 * zHyperboloid1(\yMax);
    \xH = \cHy *\yH +\cHt;
```

```
  }]
  \setlayers{-2, -1, main, 1, 2}
  \path (-2, .1*\yMax, -1.6*\yMax) pic {showCoordinates={1.15}{-1}{-2}};

  \draw[B, fill=B!50, fill opacity=.5]
  (-\xH, -\yH, -\zH) -- (\xH, \yH, -\zH)
  -- (\xH, \yH, \zH) -- (-\xH, -\yH, \zH) -- cycle;

  \begin{scope}[sor and H={name=Hyperboloid1, meridians=25, parallels=24,
      t_ini=-\yMax, t_end=\yMax, rAxis={0, 0, 0},
      cHyperplane={-1, \cHy, 0, \cHt}}]
    \path (0, 0, 0) pic[O] {rSurface and HL={above=2, below=1, actions={}}};
  \end{scope}

  \begin{scope}[sor and H={name=Hyperboloid1, meridians=25, parallels=24,
      t_ini=-\yMax, t_end=\yMax, rAxis={0, 0, 0},
      cHyperplane={1, -\cHy, 0, -\cHt}}]
    \path (0, 0, 0) pic[O] {rSurface and HL={above=-1, below=-2, actions={}}};
  \end{scope}
\end{tikzpicture}
```

8.5 Example: two interpenetrating sphere caps

This example, drawn from `merginSpheres.tex`, shows two hemispheres clipped by opposite vertical half-spaces and overlaid to create the illusion of two interpenetrating spheres.

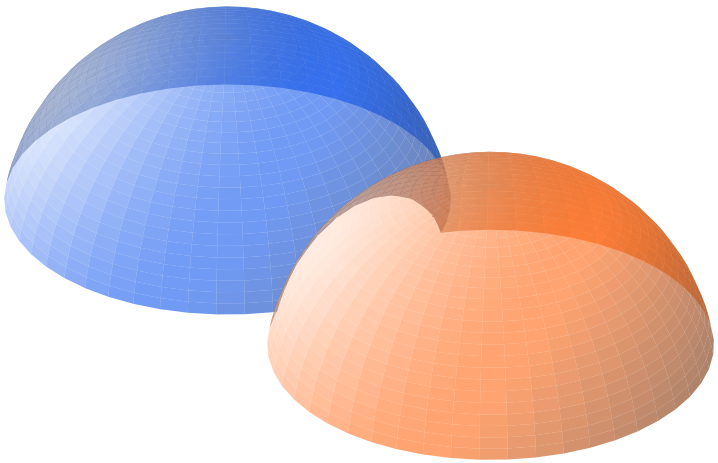

Figure 11: Two merging shere caps.

```
\tikzmath{
  real \r, \aBase, \rBase, \deltaBase, \xH, \aH;
  \r = 3;       % sphere radius
  \aBase = 10;  % parameter offset (degrees)
  \rBase = \r * cos(\aBase);
  \deltaBase = \r * sin(\aBase);
  \xH = .85*\r; % position of the clipping plane
  \aH = acos(\xH/\rBase);
}
```

```
\begin{tikzpicture}[view={47}{31}, sun={-50}{40}{.8}]
  \setlayers{-2, -1, main, 1, 2}

  %% Left sphere: clipped by x <= xH (keep x >= -xH side)
  \begin{scope}[sphereRadius=\r,
    sor and H={name=Sphere, meridians=49, parallels=27,
      t_ini=10, t_end=90, rAxis={0, 0, 0},
      cHyperplane={-1, 0, 0, \xH}}]
    \path (-\xH, -\deltaBase, 0)
    pic[B] {rSurface and HL={above=-1, below=-2,
        actions={fill opacity=.6}}};
  \end{scope}

  %% Right sphere: clipped by x >= -xH (keep x <= xH side)
  \begin{scope}[sphereRadius=\r,
    sor and H={name=Sphere, meridians=49, parallels=27,
      t_ini=10, t_end=90, rAxis={0, 0, 0},
      cHyperplane={1, 0, 0, \xH}}]
    \path (\xH, -\deltaBase, 0)
    pic[O] {rSurface and HL={above=2, below=1, actions={opacity=.6}}};
  \end{scope}
\end{tikzpicture}
```

A few things to note:

- The two spheres are shifted in opposite x-directions by `\xH` so their centres are at $(\pm$`xH`$, -$`deltaBase`$, 0)$—the two caps are "sitting" on the zy-coordinate plane.
- The left sphere uses layers -2 (back) and -1 (front hidden) while the right sphere uses 1 (hidden) and 2 (front visible). This five-layer setup ensures the interpenetration is rendered correctly: parts of each sphere that are "inside" the other appear behind it.
- The `t_ini=10` and `t_end=90` restrict the sphere to a sphere cap (a portion of the the upper hemisphere).
- The background rectangle and the circles on the cutting plane are drawn separately using `canvas is zx plane at y=0`.

9 Quick reference

9.1 Styles

`view={lon}{lat}` Set observer direction.

`sun={lon}{lat}{contrast}` Set light source.

`sor={name=N, meridians=M, parallels=P, t_ini=a, t_end=b, rAxis={ax,az,ay}}`
Precompute surface grid (no clipping).

`sor and H={..., cHyperplane={a,b,c,d}}` Precompute surface grid with half-space clipping.

9.2 Pic objects

Pic name	Clipped?	Lit?
`rSurface`	No	No
`rSurface L`	No	Yes
`partial rSurface`	No	No
`partial rSurface L`	No	Yes
`rSurface and H`	Yes	No
`rSurface and HL`	Yes	Yes
`partial rSurface and H`	Yes	No
`partial rSurface and HL`	Yes	Yes
`meridian`	single meridian line	
`parallel`	single parallel line	
`meridians`	family of meridians	
`parallels`	family of parallels	
`showCoordinates`	coordinate axes indicator	

All surface pic objects take `above=`*layer*, `below=`*layer*, `actions={...}`. The `partial` variants also take `i from=`k, `i to=`k.

9.3 Built-in curves and their parameters

Name	Parameter range	Shape keys
`Sphere`	$[-90°, 90°]$	`sphereRadius`
`Cone`	$[0, 1]$	`baseRadius`, `topRadius`, `height`
`Hyperboloid1`	$[-y_{\max}, y_{\max}]$	`z half axis`, `y half axis`
`Hyperboloid2`	$[0, z_{\max}]$	`z half axis`, `y half axis`
`Ellipsoid`	$[-90°, 90°]$	`z half axis`, `y half axis`
`Torus`	$[0°, 360°]$	`tMajorRadius`, `tMinorRadius`

9.4 Tips and common pitfalls

1. **Always declare layers** with `\setlayers` at the start of each `tikzpicture`. Forgetting this will cause pgf errors or invisible output.

2. **Layer numbering must be consistent.** Use layers $-1, \text{main}, 1$ for a single surface, and add more layers (e.g. $-2, -1, \text{main}, 1, 2$) when two surfaces overlap.

3. **The `L` variants require `sun`.** If you use `rSurface L` without setting `sun`, `contrast` is -1 and shading is silently disabled.

4. **Meridian count affects symmetry.** For a surface that should look round, use $M \geq 24$. Low values (e.g. $M = 6$ or 12) give a faceted look that can be intentional (e.g. for polyhedra).

5. **Sequential scopes only.** Since `sor` writes to global arrays, you cannot nest two `sor` scopes. Draw each surface in its own `scope[[sor=...]]` completely before starting the next.

6. **Use `tikzexternalize`.** A sphere with $M = 36$, $P = 28$ requires computing $\approx 36 \times 28 = 1008$ faces, each involving several `tikzmath` expressions. Compilation can take 10–30 seconds. Externalisation caches the result as a PDF after the first run.

7. **Half-space clipping and `t_ini`.** When using `sor and H`, make sure `t_ini` puts the first parallel safely inside the positive half-space, not on the boundary. A small offset (e.g. `t_ini=10` for a sphere) avoids degenerate boundary polygons.