BuckeyeVR Plot User's Guide

Version 1.2

Updated: January 26, 2018

Joseph R. H. Smith, Chris M. Orban, Chris D. Porter
Department of Physics, The Ohio State University

Jonathan Brown
Chemical and Bimolecular Engineering, The Ohio State University

Bart Snapp, Jim Fowler
Department of Mathematics, The Ohio State University

Contents

1 Introduction

Welcome to BuckeyeVR, a new framework for plotting parametric surfaces, curves, and vector fields on smartphones in virtual reality (VR). Visualizations can be viewed in stereoscopic VR using an Android or iOS smartphone[1] and a Google Cardboard compatible viewer, which can be purchased for a few dollars or made yourself. The app also supports a single-view mode where you can rotate the phone or swipe and pinch to zoom to change the view like with a 360° video. To downloaded the app, scan the Quick Response (QR) code in Fig. 1, search for "BuckeyeVR 3D Plot Viewer" in your device's app store, or check out the plotting website, which just requires a computer with an internet connection.

Figure 1: Scan this QR code to get the app for your device.

Section 2 gives a quick introduction to the plotting website, Section 3 provides an overview of the commands used to generate plots, and Section 4 provides several more examples.

2 Getting Started

Visualizations in BuckeyeVR are created with simple scripts that are loaded onto smartphones with QR codes. In the script, we describe a set of objects (surfaces, curves, etc.) that have certain properties (such as their color or parametric equations). For example, let's start by plotting a parametric curve. In this case let's draw a helix, which we will call `helix` (variable name), which we want to be a `parametricCurve` (type). We write

```
helix=parametricCurve
```

[1] Google cardboard apps should generally run on phones that have a gyroscope and are running Android version 4.1 or higher or iOS version 8.0 or higher https://support.google.com/cardboard/answer/6295046?hl=en.

where '`helix`' is the name of the object, which we could have also called `curve`, `h`, or `bob` (something unrelated to the actual object). Each object in the visualization must have a unique name that cannot contain equal signs or periods.

Then we set the properties of the curve where we use the syntax

```
variableName.property=X
```

where for a parametric curve we can set the bounds of `t`, the equation for the curve and properties related to how the curve is displayed (as will be expained later).

```
helix.setMinMax=t,-10,10
helix.setEquation=x,(cos(t))
helix.setEquation=y,(sin(t))
helix.setEquation=z,(.3*t)
helix.setSegmentCount=70
helix.setThickness=0.15
helix.color=red
```

Then to draw the object on the screen, or 'spawn' the object, where we say

```
helix.spawn
```

(this line must be the last for a given object).

2.1 Plotting Website

Plotting scripts can be developed and tested on the plotting website buckeyevr.osu.edu/plot and then loaded onto a smartphone with a QR code. The general layout of the site is shown in Fig. 2 and described below.

1. **Script Entry Text Box**

   The site will be preloaded with an example script in this text box to demonstrate the format of a BuckeyeVR script. The first line must be

   //ChangeFileNameHere.bvr

   where **ChangeFileNameHere** is the name of the visualization stored in the smartphone. For example if I wanted to have create a visualization called **Parabola**, the first line would read

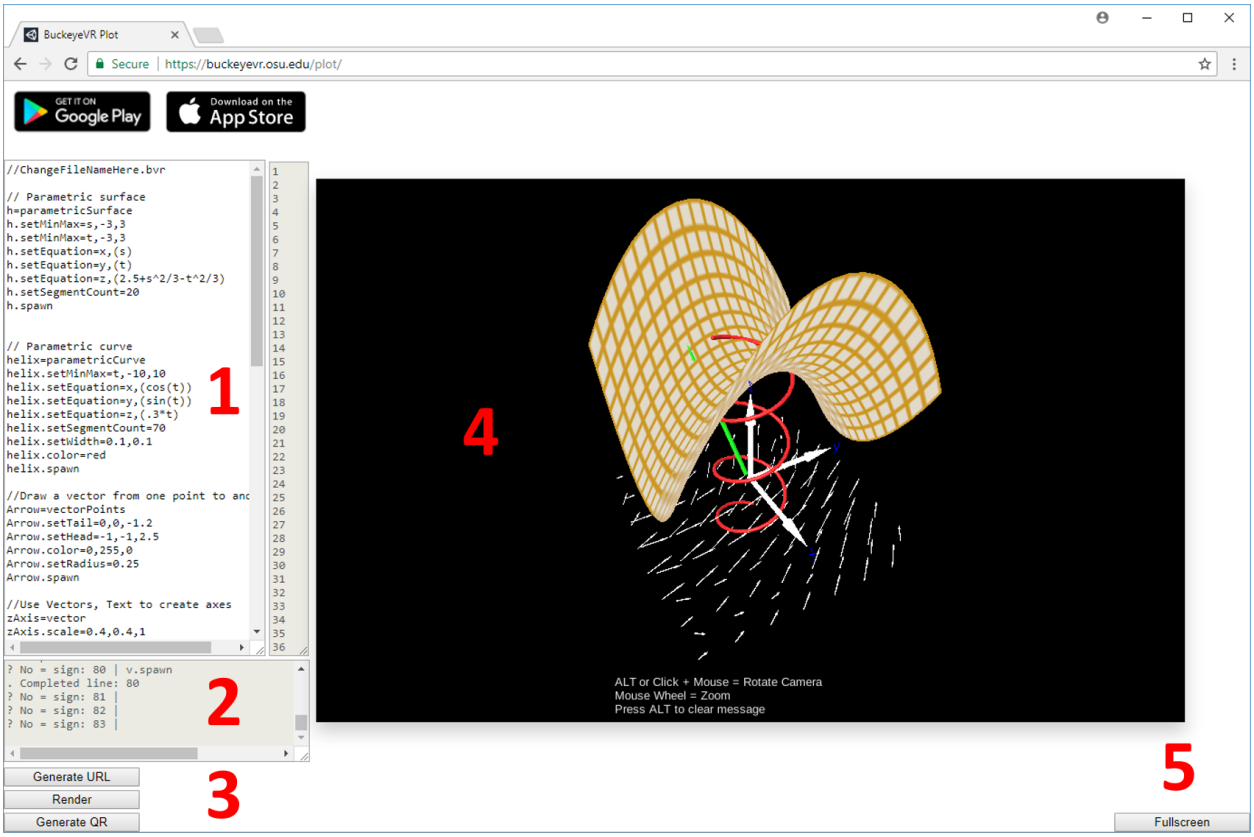


Figure 2: Layout of the plotting website (subject to change).

//Parabola.bvr

You must leave the two forward slashes (`//`) before the name and the `.bvr` extension after the name. Putting two forward slashes before a line indicates that the line is a comment, which is ignored by the interpreter.

2. **Error Detection Box**

   This box is used for basic debugging of scripts. If a visualization does not render, the output in this box 'may' be helpful for tracking down the error.

3. **Selection Buttons**

   Generate URL

   This button will generate a popup such as the one in Fig. 3, which will prompt you to press `Ctrl` (`Cmd` for Mac) and `c` to copy the given url. If you look more closely at the url, you can see that it is just the script after `https://buckeyevr.osu.edu/plot/?` (with

certain characters swapped out to be more URL-friendly). This allows you to create a link to come back and continue working on a project, or distribute it to others to view the visualization. There are a couple of important considerations with this method

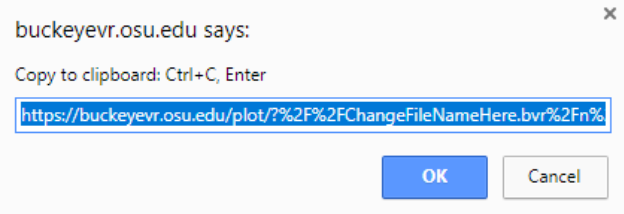


Figure 3: Example popup containing the URL that returns to the visualization.

Warning!

- In some browsers, the url in the popup has a maximum length of ∼2000 characters. It will automatically replace some of the text in the middle with '. . . '
  - If your script is getting long, you may want to check that the url retains all of the data before closing the plotting website
  - You can also just save the script in a text file and paste it into the script box to make sure it is saved.
- If you edit the script you **must** generate a new url to have the script saved.

Warning!

Render

This button will display the output of the script in the plotting window on the right.

Generate QR code

This button will convert the content in the script window to a QR code (see Section 2.2 for more information).

4. **Plot Viewer**

   The output of the script is rendered here. To rotate around the objects either click and rotate the mouse, or hold the `alt` key and move the mouse. The mouse wheel can be used to zoom in and out.

5. **Fullscreen Button**

   Press this button to enter fullscreen mode (press `ESC` to exit fullscreen mode).

2.2 QR Codes

An important feature of the BuckeyeVR plotting system is the user of QR codes to load visualizations on smartphones. The app supports two modes for scanning QR codes: **Data QR codes** where the data itself is stored in the QR code, and **URL QR codes**, where the QR code just stores the URL of a website containing the plotting script as demonstrated in Figures 4 and 5.

```
//Hyperbolic Paraboloid.bvr
h=parametricSurface
h.setMinMax=s,-3,3
h.setMinMax=t,-3,3
h.setEquation=x,(s)
h.setEquation=y,(t)
h.setEquation=z,(s^2/3-t^2/3)
h.setSegmentCount=20
h.spawn

p1=parametricCurve
p1.setMinMax=t,-3,3
p1.setEquation=x,(t)
p1.setEquation=y,(0)
p1.setEquation=z,(t^2/3)
p1.setSegmentCount=50
p1.setThickness=0.3
p1.color=blue
p1.spawn

p2=parametricCurve
p2.setMinMax=t,-3,3
p2.setEquation=x,(0)
p2.setEquation=y,(t)
p2.setEquation=z,(-1*t^2/3)
p2.setSegmentCount=50
p2.setThickness=0.3
p2.color=green
p2.spawn
```

Figure 4: Data QR code (left) and the data stored in the code (right).

https://buckeyevr.osu.edu/plot/ExampleBVRFiles/HyperbolicParaboloid.bvr

Figure 5: URL QR code (left) and the data stored in the code (right).

As scripts become more complicated, so do the Data QR codes. On the other hand, URL QR codes only point to a website and are thus often much simpler, although we have not launched an automated system to generate them. Some of the differences are summarized below.

Data QR code

- Pros
  - Generated automatically from plotting website
  - No need for internet after app is installed on device
- Cons
  - Character limit, which limits the length of scripts
    - For example, the salt crystal in Section 4.3 is too large for our data QR codes.
  - Generally more complex codes
    - Longer time for phone to scan

URL QR code

- Pros
  - Generally scanned more quickly by phone
    - Can use smaller QR codes that are still easily readable by phone
  - Can update visualization after QR code is generated
- Cons
  - Not generate automatically
  - Requires an internet connection

To create a url qr code, the `bvr` file (a text file containing the script and containing a `.bvr` extension) can be saved to a personal site, where the url **must** end in `Filename.bvr` (e.g. https://buckeyevr.osu.edu/plot/ExampleBVRFiles/NaCl.bvr) (currently this is not compatible with Google Drive, Dropbox, and similar services, although it works with GitHub as discussed in Section 2.2.1). There are a variety of free QR code generating websites and they can also be generated automatically in a variety of languages such as PYTHON or LaTeX.

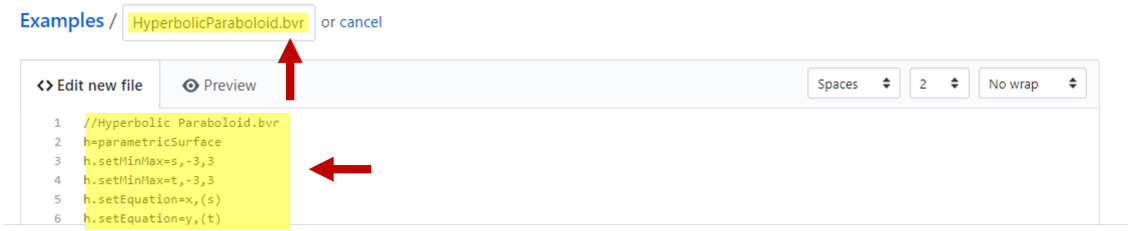
2.2.1 Using GitHub to Share Scripts

If you do not have access to a personal site to post `bvr` scripts, another option is GitHub[2] (there are plenty of source code repositories that should also work; we demonstrate a method using GitHub's web-based tools to lower the barrier to entry for users unfamiliar with git).

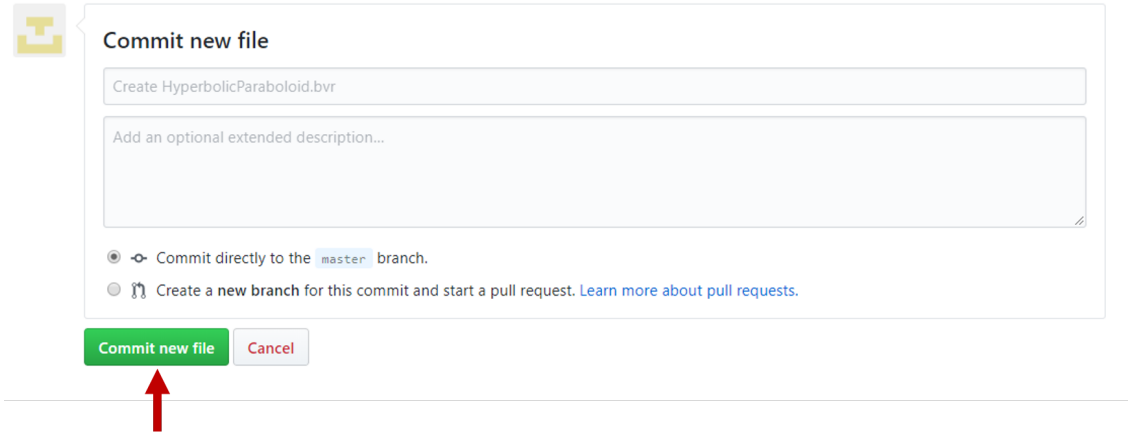
[2]Information presented here is subject to change, we have no affiliation with GitHub.

1. Create an account with GitHub,
2. Create a repository as discussed here
3. Create a new file

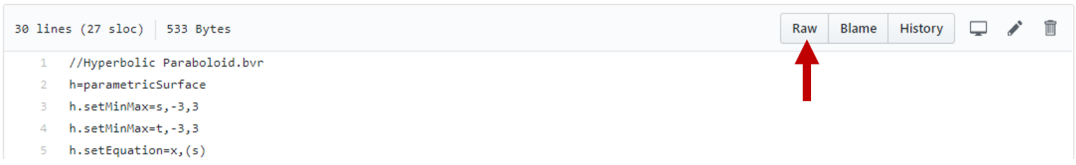
4. Choose a filename ending with `.bvr` and write the script



5. When you are done working, commit the new file



6. Look for the url of the 'raw' version, which just contains the code. Click on the file you just committed and select the raw option. It should redirect you to a page such as https://raw.githubusercontent.com/BuckeyeVR/Examples/master/HyperbolicParaboloid.bvr.



7. Create the QR code

This process is a little more involved (although in general only steps 4-7 must be repeated).

3 Plotting Options

3.1 Common Plotting Options

First we discuss several common plotting options shared by several of the plotting objects.

3.1.1 color

The color of most objects can be set with the same syntax (vector fields currently do not have this property). After an object '`varName`' has been declared, the color can be set with

```
varName.color=red
```

where the available basic colors follow.

- black
- blue
- cyan
- aqua
- gray
- grey
- green
- magenta
- fuschia
- red
- white
- yellow

Additional colors can be used with the syntax

```
varName.color=R,G,B
```

where `R`,`G`,and `B` accept values from 0 to 255.

3.1.2 Functions and Operations

For use in equations, the following functions (from `Math.f`) and operators are implemented.

- sqrt
- cos
- acos
- sin
- asin
- tan
- atan
- abs

- log
- ln
- +
- -
- /
- ∧
- %

3.1.3 spawn

After the properties of an object are set, it must be rendered to appear in the visualization. For all objects except those discussed in Section 3.5, `varName.spawn` must be called for the object.

3.2 Parametric Curves

Parametric curves have the type `parametricCurve`. The curve is parameterized with `t` and the equations are represented in Cartesian coordinates.

3.2.1 setMinMax

We may begin by settings the bounds for the parameter `t` with:

```
curve.setMinMax=t,0,3.14
```

where in this case we have $0 \leq t \leq 3.14$. We require that the first number to be smaller than the second number.

3.2.2 setEquation

The equations for `x`, `y`, and `z` are then given in terms of `t` using the syntax

```
curve.setEquation=x,(t)
curve.setEquation=y,(t)
curve.setEquation=z,(t)
```

where `t` is replaced with the desired function of `t` (including 0 or another constant to plot a 2D parametric curve). The parenthesis around the function of `t` are required. The equations and operations that can be used in `setEquation` are discussed in Section 3.1.2

3.2.3 setSegmentCount

The parametric curves are plotted as a number of linear segments that follow the path of the curve. We could represent a curve with 100 segments

```
curve.setSegmentCount=100
```

The curve can be better represented with more segments, although this can cause the visualization to render slowly. As a rule of thumb, a few hundred segments should be fine (although when there are more objects in the scene such as parametric surfaces, this may need to be reduced to mitigate lagging with the rotation of a smartphone).

Due to the current plotting scheme smaller numbers of segments are used, the curve may not exactly match the bounds in Section 3.2.1 (`setMinMax`). This can be fixed my changing the number of segments, or adjusting the bounds. For example, if you are drawing a circle with only 20 segments, you may wish to use the range $[0, 6.28]$ you have to use the bounds $[0, 6.5]$ to make sure the curve closes. This will likely be improved with further updates.

3.2.4 setThickness

The parametric curve is given some thickness when rendered in 3D space. The radius of this curve can be set using

```
curve.setThickness=0.15
```

Older versions of the code used a similar functions `setWidth`, which accepted two values (with only the first contributing to the thickness). This old methods still works, but should be replaced with `setThickness` moving forward.

3.2.5 color

See Section 3.1.1.

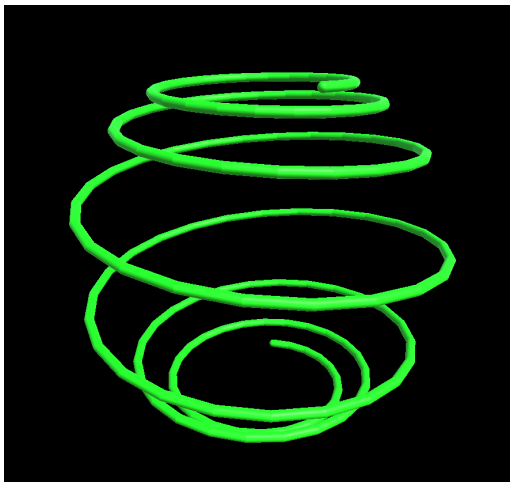
3.2.6 spawn

See Section 3.1.3.

3.2.7 Example

To demonstrate how all of these pieces work together, we will draw a spherical spiral. The code and resulting visualization are included below and can be seen on the plotting website.

```
//spiral.bvr
p=parametricCurve
p.setMinMax=t,-20,20
p.setEquation=x,(3*cos(t)*cos(atan(.1*t)))
p.setEquation=y,(3*sin(t)*cos(atan(.1*t)))
p.setEquation=z,(-3*sin(atan(.1*t)))
p.setSegmentCount=150
p.setThickness=0.15
p.color=green
p.spawn
```



3.3 Parametric Surfaces

Parametric curves are created similarly to parametric curves as discussed in Section 3.2, although the surface is parametrized with both `s` and `t`.

3.3.1 setMinMax

The range for `s` and `t` are set similarly, where

```
surf.setMinMax=s,0,3.14
surf.setMinMax=t,0,1
```

would plot `s` and `t` in the ranges $0 \leq s \leq 3.14$ and $0, \leq t \leq 1$.

3.3.2 setSegmentCount

Similar to the segments used to draw parametric curves, a set of mesh segments are used to draw parametric surfaces. For eample

```
surf.setSegmentCount=20
```

uses 20 segments in both the `s` and `t` directions. (Future versions may scale this based on range or allow it to be user defined for each parameter). Similar roundoff problems may occur as with those in Section 3.2.3.

3.3.3 setEquation

The equations for `x`, `y`, and `z` are then given in terms of `s` and `t` using the syntax

```
surf.setEquation=x,(s,t)
surf.setEquation=y,(s,t)
surf.setEquation=z,(s,t)
```

where `s,t` is replaced with the desired function of `s` and `t`. The parenthesis around the function of `t` are required. The equations and operations that can be used in `setEquation` are discussed in Section 3.1.2.

3.3.4 color

See Section 3.1.1. The color for parametric surfaces is currently not quite true to the input due to current shader settings.

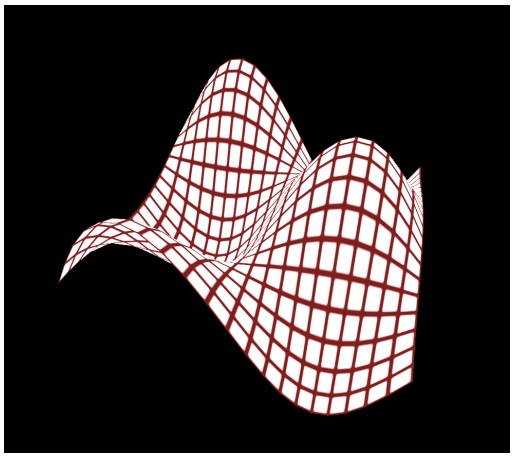
3.3.5 spawn

See Section 3.1.3.

3.3.6 Example

Here is a simple example of a surface $z = 2\sin(x)\cos(y)$, where we have used `s` and `t` to represent x and y. The script can be found on the plotting website.

```
//surface.bvr
h=parametricSurface
h.setMinMax=s,-3,3
h.setMinMax=t,-3,3
h.setEquation=x,(s)
h.setEquation=y,(t)
h.setEquation=z,(2*sin(s)*cos(t))
h.setSegmentCount=20
h.color=255,0,0
h.spawn
```



3.4 Vectors

Another feature is the ability to draw vectors from one point in space to another. These can also be used to draw axes and highlight features in the visualization. The `vectorDraw` method is used. Vectors can also be used to draw axes for plots as shown in Section 3.4.6.

3.4.1 setTail

The originating point of the vector is set using

```
vector.setTail=0,0,0
```

where $(0, 0, 0)$ is replaced with the desired coordinates.

3.4.2 setHead

The point to which the vector points is set with

```
vector.setHead=1,1,1
```

where $(1, 1, 1)$ is replaced with the desired coordinates.

3.4.3 setRadius

In order to chance the thickness of the vector, the following command is used.

```
vector.radius=0.25
```

3.4.4 color

See Section 3.1.1.

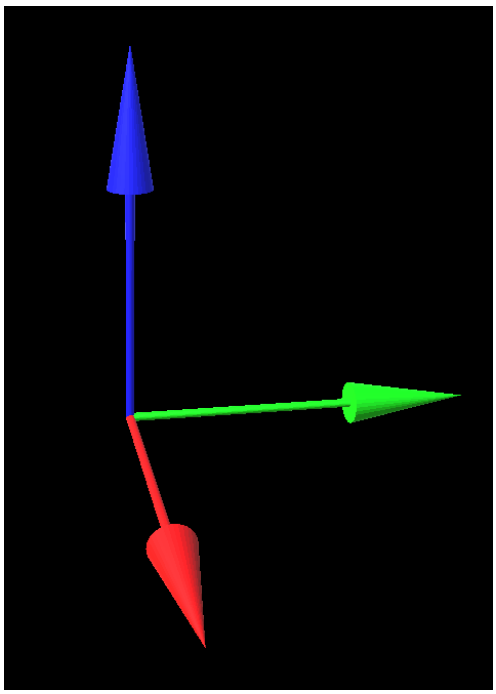
3.4.5 spawn

See Section 3.1.3.

3.4.6 Example

For example, let's draw a set of axes as demonstrated here.

```
//Axes.bvr
xAxis=vectorDraw
xAxis.setTail=0,0,0
xAxis.setHead=2,0,0
xAxis.color=red
xAxis.setRadius=0.25
xAxis.spawn

yAxis=vectorDraw
yAxis.setTail=0,0,0
yAxis.setHead=0,2,0
yAxis.color=0,255,0
yAxis.setRadius=0.25
yAxis.spawn

zAxis=vectorDraw
zAxis.setTail=0,0,0
zAxis.setHead=0,0,2
zAxis.color=blue
zAxis.setRadius=0.25
zAxis.spawn
```



3.5 Standard Objects (Cubes, spheres, text etc.)

BuckeyeVR also includes some standard shapes that can be added to visualizations including the following.

- cube
- cylinder
- sphere
- vector [3]
- plane
- capsule

These standard objects are automatically drawn when initialized (e.g. `c=cube`) and do not need to be spawned. Also 3D text can be added to the visualizations as discussed in Section 3.5.5.

[3]The vector included in these objects may not behave correctly and, in most cases `vectorDraw` should be used instead.

3.5.1 position

The default center of these objects is at the origin $(0, 0, 0)$, to chance this, one may use

```
obj.position=x,y,z
```

where (x, y, z) is the desired location of the center of the object.

3.5.2 rotation

The rotation of the objects is set using

```
obj.position=xr,yr,zr
```

where `xr`, `yr`, and `zr` are the rotations about the `x`, `y`, and `z` axes respectively in degrees.

3.5.3 scale

The size of these objects can be adjusted by scaling the object in x, y, or z. For example we could double the size of the object by scaling by two in each dimension. The scaling is performed with respect to the unrotated object.

```
obj.scale=2,2,2
```

3.5.4 color

See Section 3.1.1.

3.5.5 Text

Also `text` can be added by using the same properties and and a description

```
txt.text=Write Text Here
```

The text is represented with a 3D object and as with the other simple objects, its scale and position can be used to modify the location and size.

3.5.6 Example

These different shapes and options are demonstrated with this example

```
//objExample.bvr

h=cube
h.scale=.9,.5,1.4

s=sphere
s.position=0,0,1.3

a1=capsule
a1.position=1,0,0
a1.scale=.5,.5,.6
a1.rotation=0,40,0

a2=capsule
a2.position=-1,0,0
a2.scale=.5,.5,.6
a2.rotation=0,-40,0

l1=cylinder
l1.position=-.25,0,-1.35
l1.scale=.4,.4,.6

l2=cylinder
l2.position=.25,0,-1.35
l2.scale=.4,.4,.6

txt=text
txt.text=Hello World!
txt.color=red
txt.position=0,0,2.5

f=plane
f.position=0,0,-1.75
f.color=green
```

3.6 Vector Fields

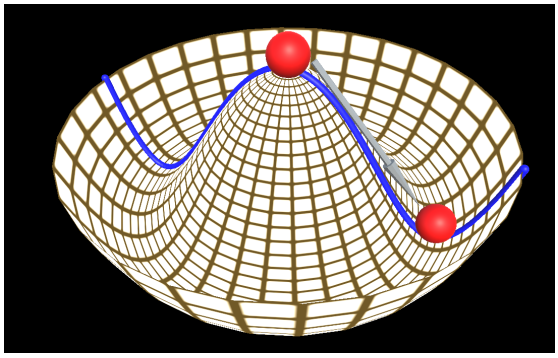
It is also possible to draw vector fields in BuckeyeVR with the `vectorField` object. It should be noted that this feature is not well tested and there are several bugs that cause this it to not behave as expected. This should be used with caution (please contact us if you are using the feature in its current form) and will be changed in future updates to correct the errors.

4 Additional Examples

4.1 Potential

Potential Example

```
//Potential Example.bvr
P=parametricSurface
P.setMinMax=s,0,6.28
P.setMinMax=t,0,1.35
P.setEquation=x,(3*sin(s)*t)
P.setEquation=y,(3*cos(s)*t)
P.setEquation=z,(3*(t^2-1)^2)
P.setSegmentCount=30
P.spawn

h=parametricCurve
h.setMinMax=t,-1.35,1.35
h.setEquation=x,(3*t)
h.setEquation=y,(0)
h.setEquation=z,(3*(t^2-1)^2)
h.setSegmentCount=90
h.setWidth=0.15,0.1
h.color=blue
h.spawn

t=sphere
t.position=0,0,3.25
t.scale=0.75,0.75,0.75
t.color=red

b=sphere
b.position=3,0,0.4
b.scale=0.75,0.75,0.75
b.color=red

v=vectorPoints
v.setTail=.25,0,3.45
v.setHead=2.4,0,1
v.setRadius=0.25
v.color=gray
v.spawn
```

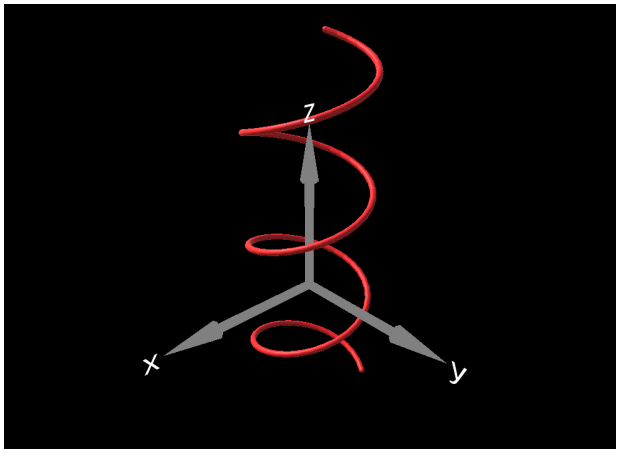


4.2 Helix Example

Helix Example

```
//Helix Example.bvr
helix=parametricCurve
helix.setMinMax=t,-10,10
helix.setEquation=x,(cos(t))
```

```
helix.setEquation=y,(sin(t))
helix.setEquation=z,(.3*t)
helix.setSegmentCount=100
helix.setWidth=0.1,0.1
helix.color=red
helix.spawn

//Use Vectors
zAxis=vector
zAxis.scale=0.4,0.4,1
zAxis.color=gray
zAxis.position=0,0,0.1
txtz=text
txtz.text=z
txtz.color=white
txtz.rotation=0,0,180
txtz.position=0,0,2

yAxis=vector
yAxis.scale=0.4,0.4,1
yAxis.color=gray
yAxis.rotation=0,90,90
yAxis.position=0,1.1,-1

txty=text
txty.text=y
txty.color=white
txty.position=0,3,0
txty.rotation=0,0,270
txty.position=0,3,-1

xAxis=vector
xAxis.scale=0.4,0.4,1.0
xAxis.color=gray
xAxis.rotation=0,90,180
xAxis.position=1.1,0,-1

txtx=text
txtx.text=x
txtx.color=white
txtx.position=3,0,-1
```



4.3 NaCl

Now we will show an example of a NaCl crystal. This script was generated using a simple `python` script. It should be noted that only a portion of the `bvr` code is included below as it is rather long. The full code can be viewed here.

Here is the python script used to generate the `bvr` file.

```
#!/usr/bin/env python
```

```
n = 0
m = 2
for x in xrange(-m,m+1):
  for y in xrange(-m,m+1):
    for z in xrange(-m,m+1):
      if (x+y+z)%2 == 0:
        print "s"+str(n)+"=sphere"
        print "s"+str(n)+".position="+str(x)+","+str(y)+","+str(z)
        print "s"+str(n)+".color=green"
        n+=1
      else:
        print "s"+str(n)+"=sphere"
        print "s"+str(n)+".position="+str(x)+","+str(y)+","+str(z)
        print "s"+str(n)+".scale=0.525,0.525,0.525"
        print "s"+str(n)+".color=red"
        n+=1
```

Here is the start of the `bvr` file, the rest can be viewed here.

```
//NaCl Crystal Example.bvr
s0=sphere
s0.position=-2,-2,-2
s0.color=green
s1=sphere
s1.position=-2,-2,-1
s1.scale=0.525,0.525,0.525
s1.color=red
s2=sphere
s2.position=-2,-2,0
       ...
```