## #1 - Introduction

Hey everyone! This is the first video that I’m making about goopylib to introduce to you what it is all about. Goopylib is a graphics library written in Python and it can be used to make games, GUIs, animations, and a lot more! I started working on it about 2 years ago when I began editing John Zelle’s graphics.py to add more functionality. Since then, it’s grown into something absolutely amazing that even I use to make my own applications.

This is goopylib.

Here’s how it’s organized. Goopylib is currently divided into 5 subpackages: objects, math, sound, applications, & physics. Together, they provide functionality that no other Python library can.

The objects subpackage is perhaps the most important, it contains everything you would need to build a responsive GUI or a beautiful game.

The math subpackage is one I am particularly proud of and it’s been an amazing journey developing it. If you need functions used in a lot of computer graphics, this is it.

And you know what’s great? It’s being actively developed – just by me for now, but hopefully, other contributors will join in time.

The sound subpackage is planned to have all sorts of tools required to bring your programs to life. This doesn’t just include the ability to play sounds on multiple channels with volume controls, but also manipulate sounds – add reverb, distortion & compression, delay, and more with just a few lines of code.

It also has a rigid-body physics engine planned in the future to allow you to incorporate gravity & collisions into your projects too.

Wanna know more? Well, I’ve worked quite a bit on optimization, so it should run pretty fast. I clocked about 40,000 frames per second on a test Pac-Man game. There are very few dependencies – only pillow & NumPy. It’s also entirely a passion project! I’ve been the sole worker on this for 2 years now.

Goopylib is incredibly pretty simple & intuitive to use too! Check this out.

Create a Window

Draw an Image to it

Find the location of a mouse click

And check if the Image was clicked

So, if you’re interested in creating graphical apps in Python with simple code and great support, give Goopy a try! And if you have any feedback or suggestions, make sure to either comment it down below, or head over to the goopylib GitHub page and post an issue – the GitHub page is also where you can find the official documentation.

## 2# - Creating a Window

Hello! In the second video about goopylib, I’d like to talk to you about windows and how you create them. A window is the most fundamental part of a graphical application and as such, knowing how to create one is incredibly important and simple.

First, let’s start by importing goopylib into our program. There are a few ways you can do this. The most basic is to simply type `from goopylib.imports import \*`, this will take everything in the library from all the subpackages and import it into your file. Alternatively, if you would only like to use the window, you could just say `from goopylib.Window import Window`.

Great! Now we have the Window class to work with.

Next, we’ll create an instance of a window itself. Let’s have it defined to a variable called `window`. `window = Window()`. And here, we can give the window some important attributes. The first will be its title: `title=”Test Window”`. We can also specify its dimensions: `width=700, height=700`. This should give us a square window.

Let’s try running the program. You’ll see that we get no window at all! This is because the program creates a window and then just ends, closing. So, we’ll add in what is known as the main loop. It’s simply just a while loop that runs while the window is open. `while window.is\_open():`. We’ll leave it empty for now.

If we were to run this now, it works! We have a window! While we have this program open, why don’t we try drawing something to the window? Import the Circle & Point classes, then we’ll draw one to the window. Don’t worry about what this means, for now, I’ll go into detail in future videos.

You’ll notice that the window doesn’t actually get a circle drawn to it. This is because every Window needs to be updated – manually. You could just do this by putting an update after you draw the circle, but I prefer doing it inside the main loop. `window.update()`.

And there we have it! It’s that simple. Goopylib provides many ways of customizing windows too and Ill mention just a few of them in this video. If you need detailed documentation, go to the link in the description.

You have already seen how to add a title to the window and set its width & height. We can make the window manually resizable by the user by using the `resizable\_width` and `resizable\_height` arguments. Providing the `min\_width`, `min\_height`, `max\_width`, and `max\_height` arguments will also allow you to customize the maximum and minimum sizes the user can expand or shrink the window to.

By default, the window appears on the top-right corner of the screen, this can be changed by providing the `x\_pos` and `y\_pos` arguments. To set the background colour of a window, use `bk\_colour` and assign it to a goopylib Colour Object. If you want to specify a custom icon for your window, you can provide a file path to a .ico file.

And those are the important ones you need to worry about. All these attributes come with getter and setter functions too so that you can edit them after the Window has been created.

This just scratches the surface of the Window class – there is so much more you can do with it which I hope to get to in future videos. If you have any questions, make sure to post them in the video description or the issues page inside GitHub.

## Bézier Curves #1 – An Introduction

1. **Introduction**

[1] If you’ve ever used vector graphics software like Adobe Illustrator or Adobe Animate, you’re probably familiar with the curved line tool – you’ve already worked with these things known as Bézier curves.

[2] Heck, even if you’ve used video editing software like Premiere Pro, you might know about Bézier Animations.

[3] Bézier curves are used to create, well, curves between a set of points and they’re a really neat thing.

1. **What are Bézier Curves?**

[1] Say you’re given 2 points: how would you construct a curve between them? Most of you might intuitively want to draw a line like this – and while that’s fine, even a straight line between the points could be considered a *curve*.

[2] This changes when we add a 3rd point – most people would draw a curve like this – something that would be more comparable to an interpolation between points.

[3] Here is how a Bézier Curve created using 3 control points would look. The curve simply uses these *control points* as guides for constructing the curve with only the start and end points being part of the curve itself. Here’s one you might be more familiar with – A Bézier Curve between 4 points. It’s the same one you use to manipulate lines in vector graphics software.

[4] Each of these curves has a different *degree* which visually translates to the number of direction changes possible in the curve.

[5] A *linear* Bézier Curve is simply a straight line – it has a degree of 1 and 0 directional changes possible. A *quadratic* Bézier Curve has a degree of 2 and can have at most a single direction change. Similarly, a 3rd-degree Bézier curve is known as a *cubic* curve and can have 2 direction changes.

[6] To generalize, an *d* degree Bézier Curve is defined by *d + 1* control points and can have *d – 1* changes in direction.

1. **The Logic behind Bézier Curves**

[1] Alright, the big question: ‘How do Bézier Curves work!’. They’re essentially just a bunch of linear interpolations put together. What are linear interpolations you ask?

[2] A linear interpolation is a way of constucting a straight line between points. Let’s create a variable and call it *t. t* will have a value between 0 and 1 which you can think of as a percentage representing ‘How far into the line the interpolation is’.

[3] At *t=0* the interpolation is just starting – it’s equal to the first point. At *t=0.25* you’re about a quarter of the way through the line. *t=0.5* represents the midway point and *t=1* is where the interpolation ends – it’s equal to the last point.

[4] A first degree Bézier Curve is the same as a linear interpolation between 2 points. But let’s tackle something a bit more interesting. What’s the logic behind creating a quadratic Bézier Curve?

[5] All you need to do is take the ‘linear interpolation of the linear interpolation’. What does this mean? Let’s start our parameter *t* at 0 again. Everytime *t* increases, we will linearly interpolate between the first and second point and the second and last point. So, in the end we get a shape like this – it’s not quite a Bézier Curve yet. But now, what if every time *t* increases, we ALSO form a linear interpolation between the linear interpolations of the control points?

[6] Something like this. At *t=0.5* our linear interpolation between the control points is at the halfway mark. Let’s use these 2 halfway marks to create another linear interpolation and since *t=0.5* this interpolation will be at the halfway mark too. Doing this for every value of *t* will form a smooth Bézier Curve!

[7] This is the principle that applies any number of control points. Form the first layer of linear interpolations, interpolate between that first layer to form a second layer, and keep doing that for every layer after that until you you only have 1 point left. Putting together those points for every value of *t* will give you a Bézier Curve.

1. **Conclusion**

And that about covers it for this video – I hope it’s helped you understand Bézier Curves – what they are and what they do. If you have any questions, post them in the comment section and I’ll try responding as quick as I can. Make sure to like the video and subscribe to the channel if you aren’t already.

I’m planning to release another video that delves a bit deeper and explains the mathematics of these curves, so make sure to keep an eye out for that.

**The Mathematics of Bézier Curves**

**Implementing Bézier Curves in Python**

### Possible Background Music Pieces:

1. Tchaikovsky Piano Trio, Mvt II
2. Chopin Nocturne No 2 Op. 9 in Eb Major
3. **Chopin Nocturne No. 3 Op. 9 in B Major - #1 Introduction**
4. **Chopin Nocturne No. 1 Op. 15 in F Major - #2 The Window**
5. Chopin Nocturne No.1 Op. 32 in B Major
6. Chopin Nocturne No. 2 Op. 62 in E Major
7. **Debussy Arabesque No. 1 – Bézier Curve #1**