Pre-Class Warmup

- Today we are going to be looking at Blender3D
  - Blender3D has gone through quite an evolution!
    - I remember the 1.0 interface long ago!
    - Blender3D was first released in 1994!
- (For fun) Here's a picture of myself with Ton Roosendaal at SIGGRAPH 2022

Tufts UNIVERSITY

Getting Started with Scripting in Python

with Mike Shah

13:30 - 14:30 Sun, Feb 11, 2024

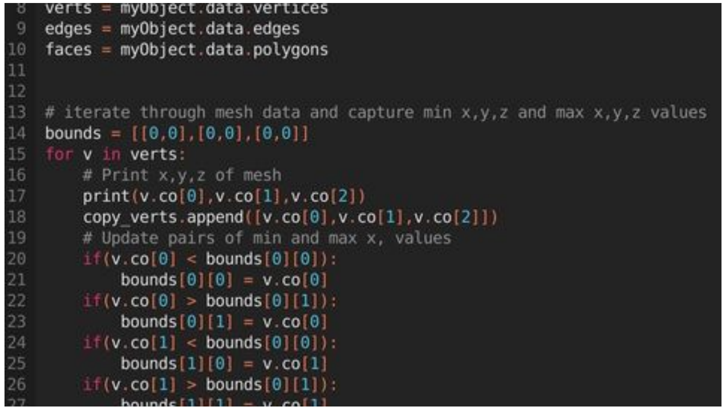
50 minutes | Introductory Audience

Social: @MichaelShah
Web: mshah.io
Courses: courses.mshah.io
YouTube:
www.youtube.com/c/MikeShah

Abstract

Note: Today's talk is based on my Blender3D Conference talk

Blender 3D is a powerful tool for 3D modeling, animation, rigging, texturing, drawing, vfx, and more -- but what happens when a feature is not available in your respective domain? Good news -- you can create it yourself! In this talk, I will be showing beginners how they can get started creating their first add-on to the Blender 3D ecosystem using Python. This talk will show you how to get started with the scripting interface for artists with minimal programming experience, or programmers who want to write tools that integrate into the Blender 3D ecosystem. Folks will leave this presentation understanding how to write, package, and find more information to develop awesome scripts where they need!



```
verts = myObject.data.vertices
edges = myObject.data.edges
faces = myObject.data.polygons


# iterate through mesh data and capture min x,y,z and max x,y,z values
bounds = [[0,0],[0,0],[0,0]]
for v in verts:
    # Print x,y,z of mesh
    print(v.co[0],v.co[1],v.co[2])
    copy_verts.append([v.co[0],v.co[1],v.co[2]])
    # Update pairs of min and max x, values
    if(v.co[0] < bounds[0][0]):
        bounds[0][0] = v.co[0]
    if(v.co[0] > bounds[0][1]):
        bounds[0][1] = v.co[0]
    if(v.co[1] < bounds[0][0]):
        bounds[1][0] = v.co[1]
    if(v.co[1] > bounds[0][1]):
        bounds[1][1] = v.co[1]
```

BCON 2023

Link to Talk:
https://www.youtube.com/watch?v=wWTAQP7-ZUQ&

Warning -- this talk may take you on a journey of spending even more time using Blender 3D to create awesome creations.

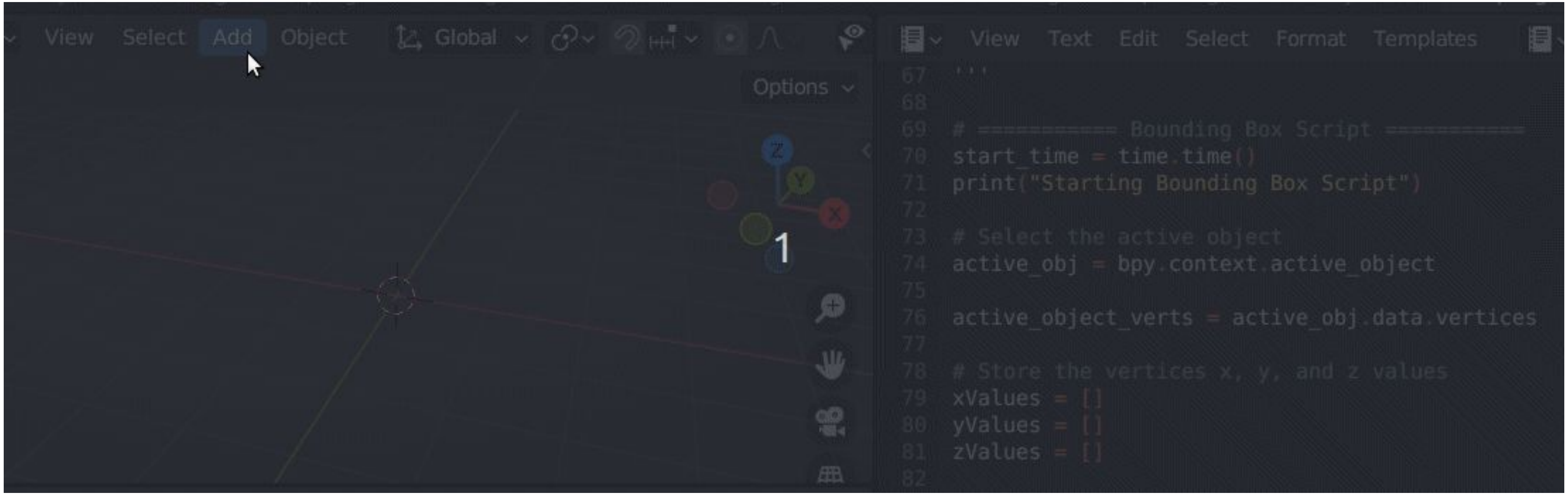
E	Rated 'E' For Everyone!
	(Yup, let's continue to make Blender3D fun for everyone involved)

Here is what we are creating!

(So you know if you should stick around or hop into another session)

Result of Today’s Presentation

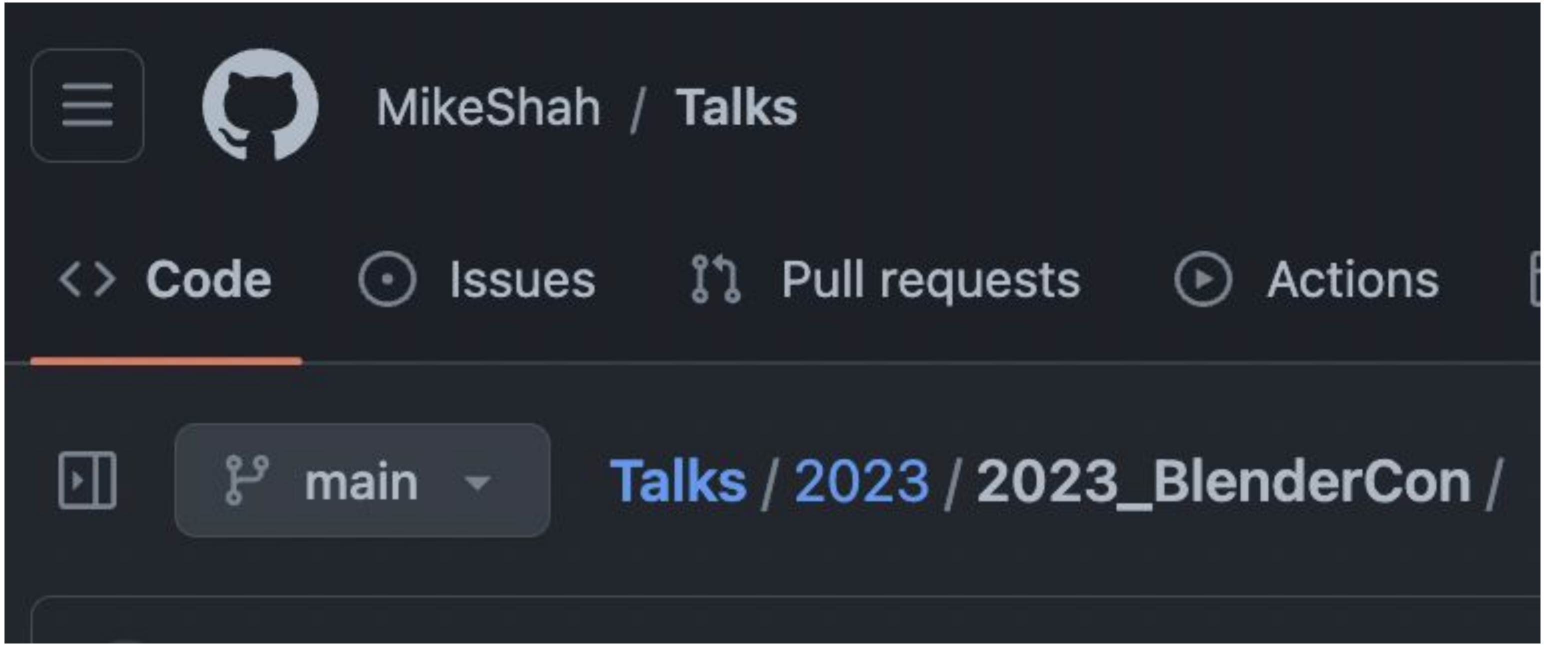
- Creating a Bounding Box programmatically in Python



```
'''


# ========== Bounding Box Script ==========
start_time = time.time()
print("Starting Bounding Box Script")

# Select the active object
active_obj = bpy.context.active_object

active_object_verts = active_obj.data.vertices

# Store the vertices x, y, and z values
xValues = []
yValues = []
zValues = []

```

Code for the talk (or Google my name and find talk listed on website)

- Located here: https://github.com/MikeShah/Talks/tree/main/2023/2023_BlenderCon



Your Tour Guide for Today
by Mike Shah

- Associate Teaching Professor at Northeastern University in Boston, Massachusetts.
  - I **love** teaching: courses in computer systems, computer graphics, geometry, and game engine development.
  - My research is divided into computer graphics (geometry) and software engineering (software analysis and visualization tools).
- I do consulting and technical training on modern C++, DLang, Concurrency, OpenGL, and Vulkan projects
  - Usually graphics or games related -- e.g. Building 3D application plugins
- Outside of work: guitar, running/weights, traveling and cooking are fun to talk about

Web
www.mshah.io
YouTube
https://www.youtube.com/c/MikeShah
Non-Academic Courses
courses.mshah.io
Conference Talks
http://tinyurl.com/mike-talks

About your instructor (3/6)

- I started writing video games around middle school.
- Shortly after I wrote programming tutorials for a gaming magazine.
- Went to undergraduate at The Ohio State University for Computer Science. Did quite a bit of research in gaming and medical visualization.
- Then wound up at Tufts to do my Masters and Ph.D. degree.
- Primarily interested in HCI, computer graphics and performance. That’s also where my professional experience lies.

About your instructor (4/6)

4D Pipeline®
DEFINE ▮ DESIGN ▮ DEVELOP ▮ DEPLOY

intel®

Ohio Supercomputer Center

Tufts
UNIVERSITY

THE OHIO STATE UNIVERSITY

Harvard Extension School
HARVARD DIVISION OF CONTINUING EDUCATION

BOSTON MEDICAL CENTER
EXCEPTIONAL CARE. WITHOUT EXCEPTION.

OHSU
OREGON HEALTH & SCIENCE UNIVERSITY

About y

Places where I worked on something “graphicsy” or performance related (all of them)

Pipeline

DEVELOP ▮ DEPLOY

Ohio Supercomputer Center

The Ohio State University

Tufts University

intel

Harvard Extension School

HARVARD DIVISION OF CONTINUING EDUCATION

Boston Medical Center

EXCEPTIONAL CARE. WITHOUT EXCEPTION.

Oregon Health & Science University

OHSU

About your instructor (3/3)

4D Pipeline®

DEFINE ▮ DESIGN ▮ DEVELOP ▮ DEPLOY

intel®

Ohio Supercomputer Center

NORTHEASTERN

HUSKIES

OREGON HEALTH & SCIENCE UNIVERSITY

OHSU

Your Tour Guide for Today -- Double Jumbo!

by Mike Shah

Halligan Hall days...

Graduation was like yesterday!

Grad Council

Granoff -- excellent study spot with some CS folks

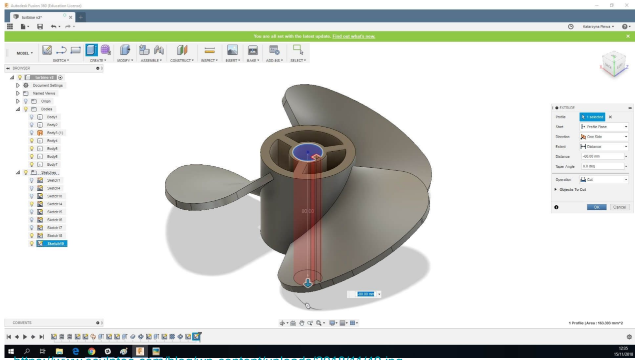
7 years of no air conditioning

3D Modeling Software

Motivation

- Today we are going to look at a specific 3D software that many use in industry
- But there exist many different software packages you can use -- so the same skills apply

https://top3dshop.com/blog/best-free-and-paid-3d-modeling-software



https://www.sculpteo.com/blog/wp-content/uploads/2018/11/40.jpg

State of the Art

- Currently (as of this writing), in your instructors opinion, Maya3D is the long standing industry standard if I had to choose just one software.
  - The race is however is incredibly tight, so I want to show you several 3D software packages.
  - Through the next several slides I will show you some 3D packages you should be familiar with in various industries
    - This is not an exhaustive list by any means, but a sampling of packages and a brief summary of their main features
    - Note: I have used some of these tools, but not all of them -- some of you may have more experience (or otherwise have worked at these companies!)

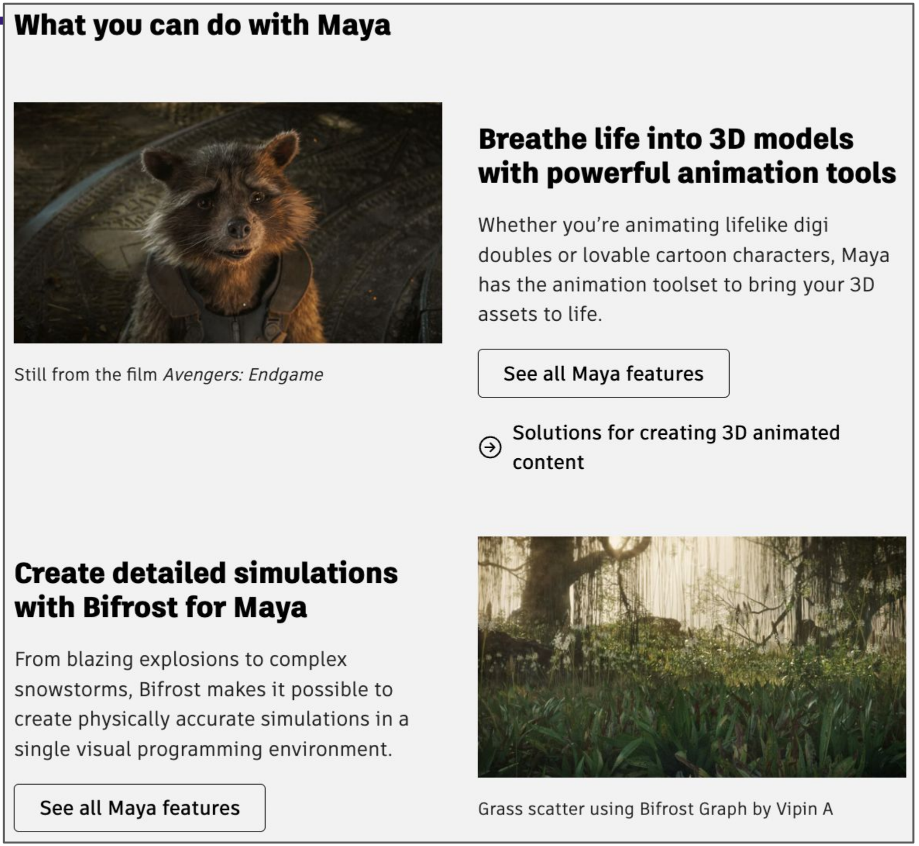
3D Modeling Software - Maya3D

“Maya is professional 3D software for creating realistic characters and blockbuster-worthy effects.”

- *Bring believable characters to life with engaging animation tools.*
- *Shape 3D objects and scenes with intuitive modeling tools.*
- *Create realistic effects—from explosions to cloth simulation*

  .

- Free Student licenses are available
- Usually a new edition every year



What you can do with Maya

Still from the film *Avengers: Endgame*

Breathe life into 3D models with powerful animation tools

Whether you’re animating lifelike digi doubles or lovable cartoon characters, Maya has the animation toolset to bring your 3D assets to life.

See all Maya features

Solutions for creating 3D animated content

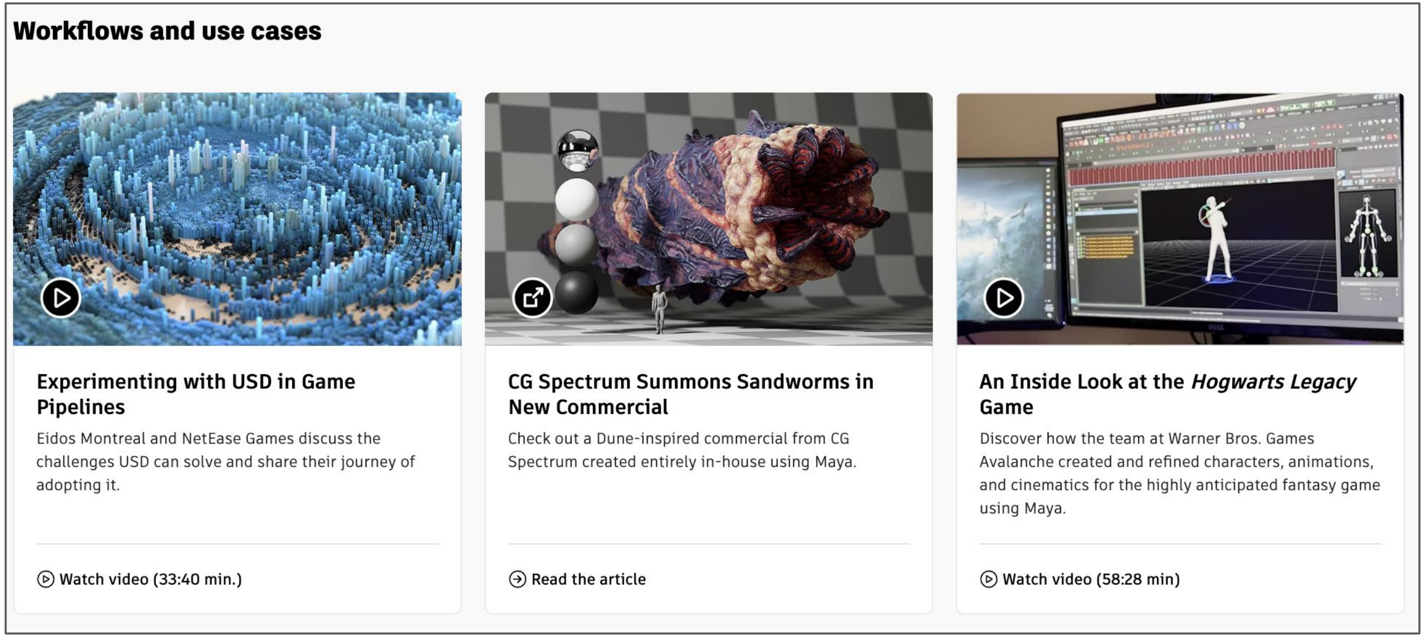
Create detailed simulations with Bifrost for Maya

From blazing explosions to complex snowstorms, Bifrost makes it possible to create physically accurate simulations in a single visual programming environment.

See all Maya features

Grass scatter using Bifrost Graph by Vipin A

- https://www.autodesk.com/products/maya/overview?term=1-YEAR&tab=subscription
- Maya is used everywhere from motion pictures to AAA game studios



Workflows and use cases

Experimenting with USD in Game Pipelines
Eidos Montreal and NetEase Games discuss the challenges USD can solve and share their journey of adopting it.
Watch video (33:40 min.)

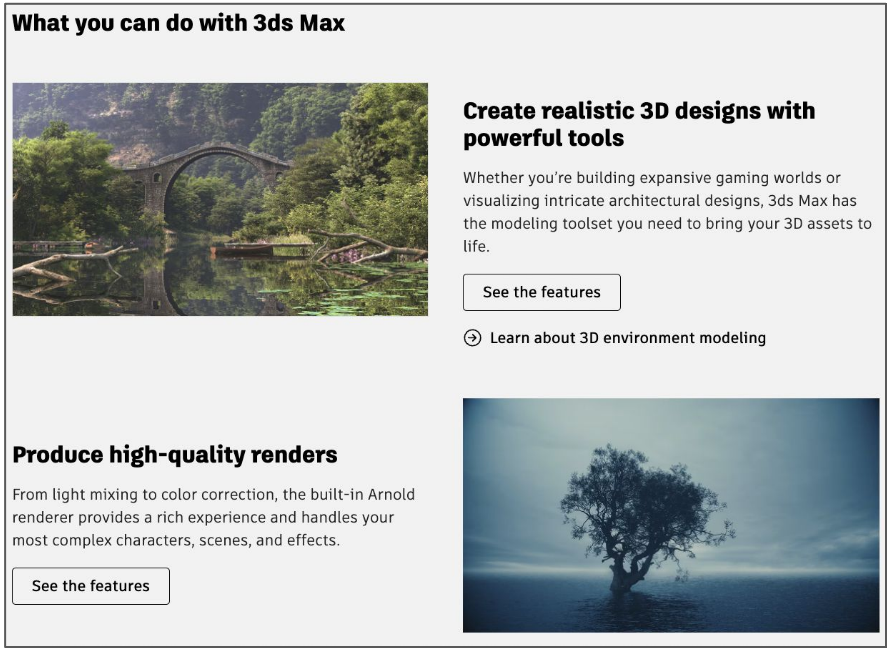
CG Spectrum Summons Sandworms in New Commercial
Check out a Dune-inspired commercial from CG Spectrum created entirely in-house using Maya.
Read the article

An Inside Look at the *Hogwarts Legacy* Game
Discover how the team at Warner Bros. Games Avalanche created and refined characters, animations, and cinematics for the highly anticipated fantasy game using Maya.
Watch video (58:28 min)

3D Modeling Software - 3ds Max

- Autodesk 3ds Max® professional 3D modeling, rendering, and animation software enables you to create expansive worlds and premium designs.
- Breathe life into environments and landscapes with robust modeling tools.
- Create finely detailed designs and props with intuitive texturing and shading tools.
- Iterate and produce professional-grade renders with full artistic control.

- Free Student licenses are available
- Usually a new edition every year



What you can do with 3ds Max

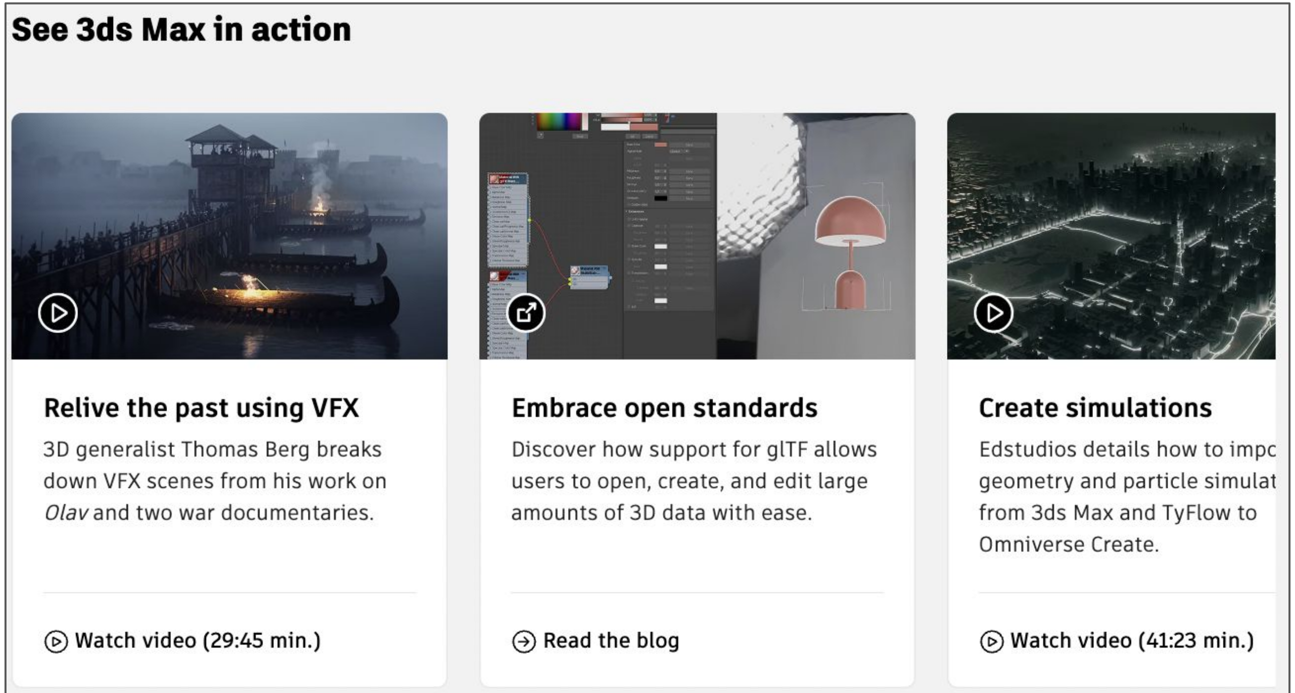
Create realistic 3D designs with powerful tools

Whether you're building expansive gaming worlds or visualizing intricate architectural designs, 3ds Max has the modeling toolset you need to bring your 3D assets to life.

Produce high-quality renders

From light mixing to color correction, the built-in Arnold renderer provides a rich experience and handles your most complex characters, scenes, and effects.

- https://www.autodesk.com/products/3ds-max/overview?term=1-YEAR&tab=subscription



See 3ds Max in action

Relive the past using VFX

3D generalist Thomas Berg breaks down VFX scenes from his work on *Olav* and two war documentaries.

Watch video (29:45 min.)

Embrace open standards

Discover how support for glTF allows users to open, create, and edit large amounts of 3D data with ease.

Read the blog

Create simulations

Edstudios details how to impc geometry and particle simulat from 3ds Max and TyFlow to Omniverse Create.

Watch video (41:23 min.)

(Aside)

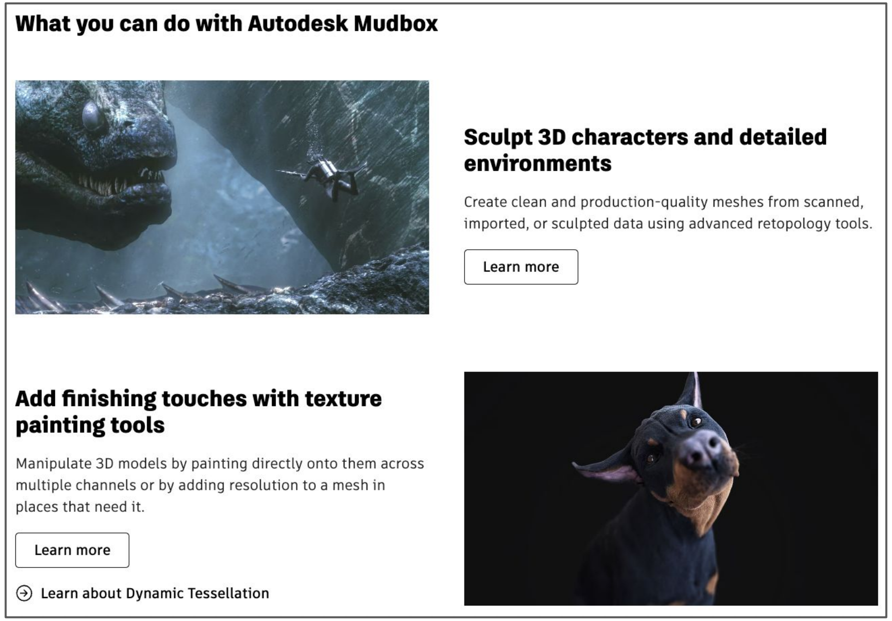
- Observe Autodesk owns both Maya and 3ds Max
  - Maya3D was acquired by Autodesk
  - 3ds Max and Maya largely can do similar things.

3D Modeling Software - Mudbox

Mudbox® software is a digital 3D sculpting and painting tool for creating detailed characters and environments with beautiful textures.

- *Bring 3D designs to life with intuitive sculpting tools.*
- *Add final touches to models with flexible texture painting capabilities.*
- *Work faster with an easy-to-use layer-based workflow*

- This was a company acquired by Autodesk as well


What you can do with Autodesk Mudbox

Sculpt 3D characters and detailed environments

Create clean and production-quality meshes from scanned, imported, or sculpted data using advanced retopology tools.

Add finishing touches with texture painting tools

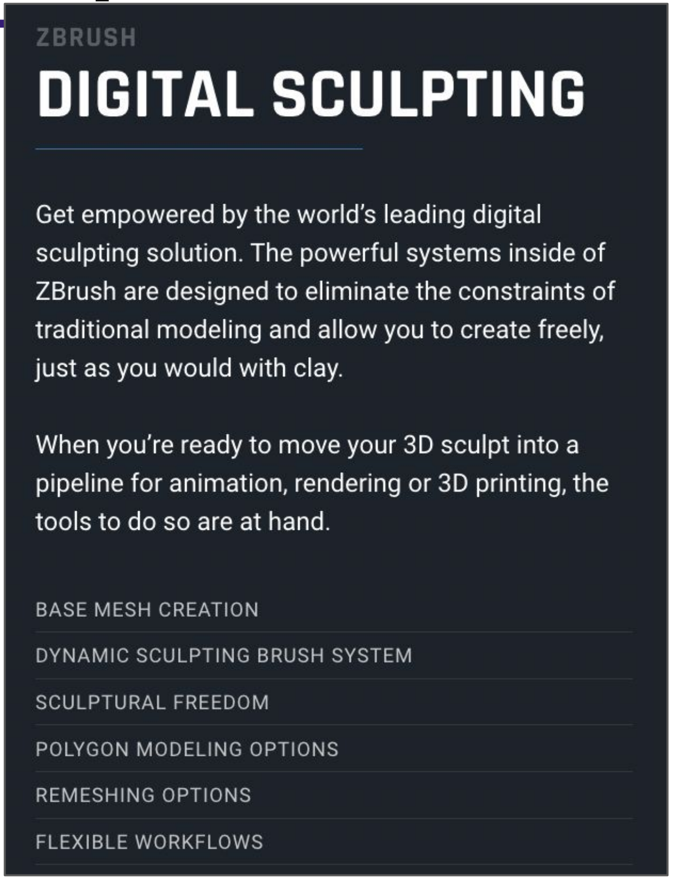
Manipulate 3D models by painting directly onto them across multiple channels or by adding resolution to a mesh in places that need it.

3D Modeling Software - Mudbox [website]

Mudbox® software is a digital 3D sculpting and painting tool for creating detailed characters and environments with beautiful textures.

- *Bring 3D designs to life with intuitive sculpting tools.*
- *Add final touches to models with flexible texture painting capabilities.*
- *Work faster with an easy-to-use layer-based workflow*

- Free Student licenses are available
- Usually a new edition every year



ZBRUSH

DIGITAL SCULPTING

Get empowered by the world's leading digital sculpting solution. The powerful systems inside of ZBrush are designed to eliminate the constraints of traditional modeling and allow you to create freely, just as you would with clay.

When you're ready to move your 3D sculpt into a pipeline for animation, rendering or 3D printing, the tools to do so are at hand.

BASE MESH CREATION

DYNAMIC SCULPTING BRUSH SYSTEM

SCULPTURAL FREEDOM

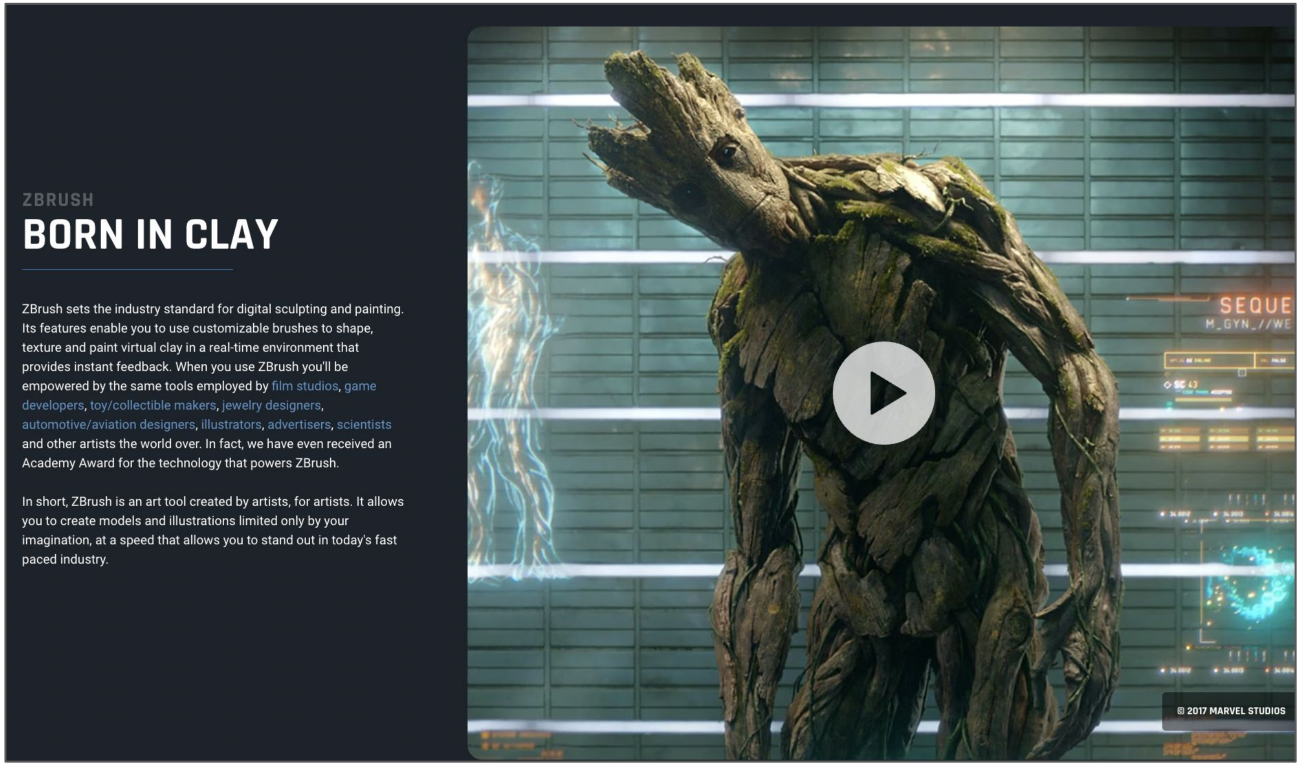
POLYGON MODELING OPTIONS

REMESHING OPTIONS

FLEXIBLE WORKFLOWS

3D Modeling Software - Zbrush [website]

- ZBrush is primarily a sculpting and painting tool
- Many motion pictures utilize ZBrush



ZBRUSH

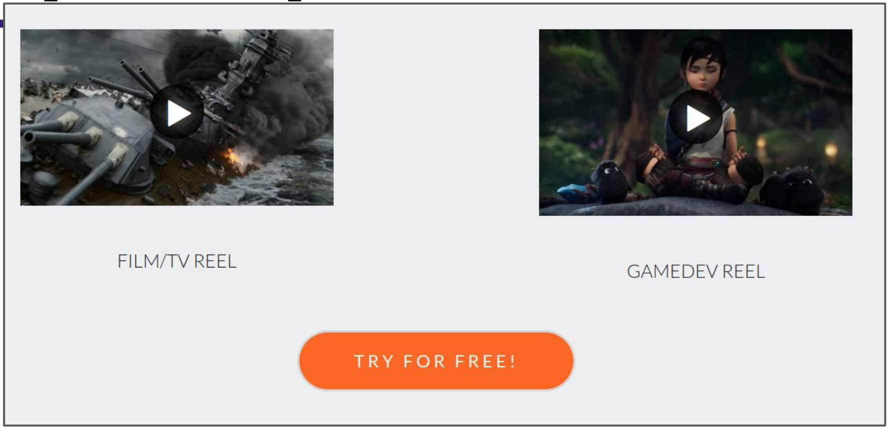
BORN IN CLAY

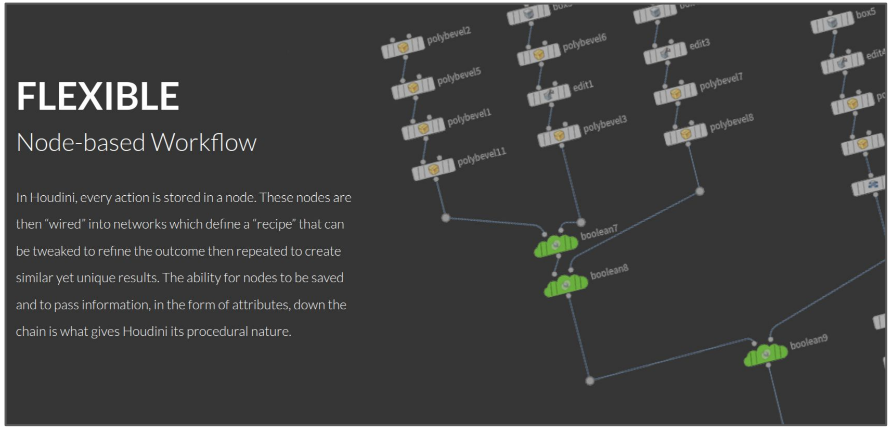
ZBrush sets the industry standard for digital sculpting and painting. Its features enable you to use customizable brushes to shape, texture and paint virtual clay in a real-time environment that provides instant feedback. When you use ZBrush you'll be empowered by the same tools employed by film studios, game developers, toy/collectible makers, jewelry designers, automotive/aviation designers, illustrators, advertisers, scientists and other artists the world over. In fact, we have even received an Academy Award for the technology that powers ZBrush.

In short, ZBrush is an art tool created by artists, for artists. It allows you to create models and illustrations limited only by your imagination, at a speed that allows you to stand out in today's fast paced industry.

3D Modeling Software - Houdini [website]

- *Houdini is built from the ground up to be a procedural system that empowers artists to work freely, create multiple iterations and rapidly share workflows with colleagues.*
- Free trials and indie versions availble





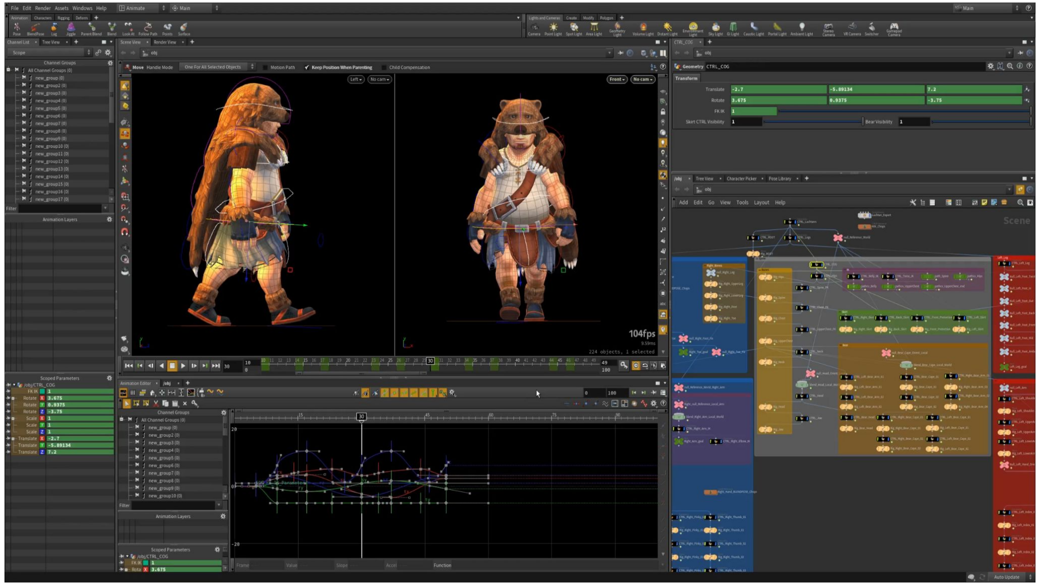
FLEXIBLE

Node-based Workflow

In Houdini, every action is stored in a node. These nodes are then "wired" into networks which define a "recipe" that can be tweaked to refine the outcome then repeated to create similar yet unique results. The ability for nodes to be saved and to pass information, in the form of attributes, down the chain is what gives Houdini its procedural nature.

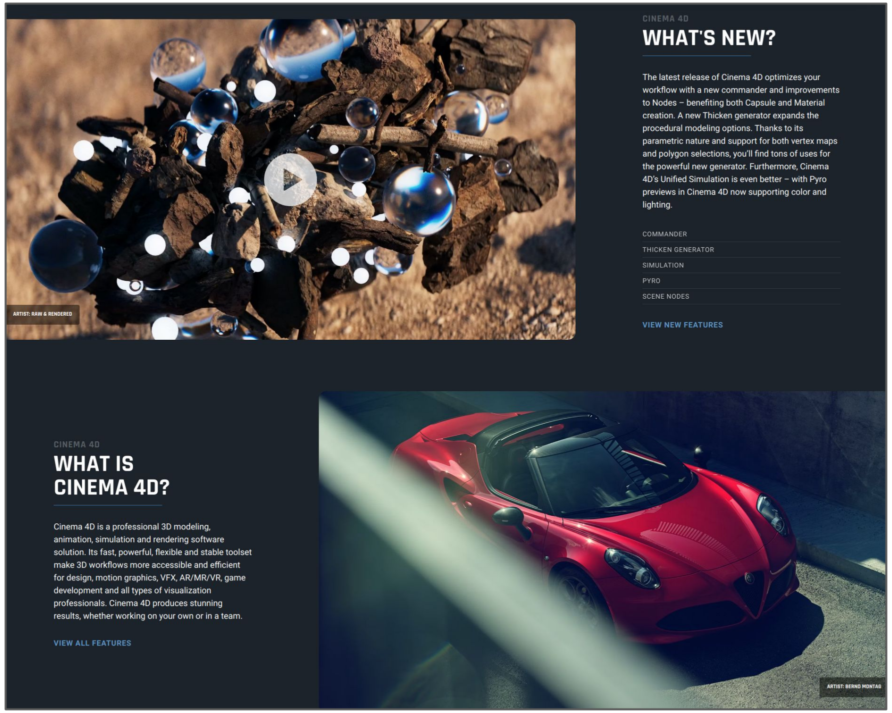
3D Modeling Software - Houdini [website]

- Houdini known for their animation and simulation tools
  a. (Or at least your instructors knowledge of the tool is for that purpose)
  b. That said, this is a full modeling, animation, texturing, VFX, package.



3D Modeling Software - Cinema 4D [website]

- *Cinema 4D is a professional 3D modeling, animation, simulation and rendering software solution. Its fast, powerful, flexible and stable toolset make 3D workflows more accessible and efficient for design, motion graphics, VFX, AR/MR/VR, game development and all types of visualization professionals. Cinema 4D produces stunning results, whether working on your own or in a team.*



CINEMA 4D

WHAT'S NEW?

The latest release of Cinema 4D optimizes your workflow with a new commander and improvements to Nodes – benefiting both Capsule and Material creation. A new Thicken generator expands the procedural modeling options. Thanks to its parametric nature and support for both vertex maps and polygon selections, you'll find tons of uses for the powerful new generator. Furthermore, Cinema 4D's Unified Simulation is even better – with Pyro previews in Cinema 4D now supporting color and lighting.

COMMANDER
THICKEN GENERATOR
SIMULATION
PYRO
SCENE NODES

VIEW NEW FEATURES

CINEMA 4D

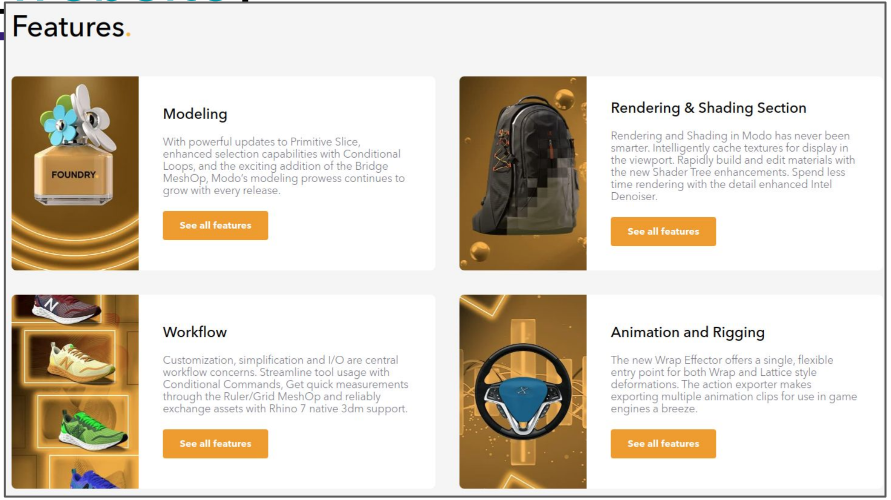
WHAT IS CINEMA 4D?

Cinema 4D is a professional 3D modeling, animation, simulation and rendering software solution. Its fast, powerful, flexible and stable toolset make 3D workflows more accessible and efficient for design, motion graphics, VFX, AR/MR/VR, game development and all types of visualization professionals. Cinema 4D produces stunning results, whether working on your own or in a team.

VIEW ALL FEATURES

3D Modeling Software - Modo [website]

- *Modo's powerful and flexible 3D modeling, animation, texturing and rendering toolset empowers artists to explore and develop ideas without jumping through technical hoops. Modo® is your starting point for creative exploration.*



Features.

Modeling

With powerful updates to Primitive Slice, enhanced selection capabilities with Conditional Loops, and the exciting addition of the Bridge MeshOp, Modo's modeling prowess continues to grow with every release.

Rendering & Shading Section

Rendering and Shading in Modo has never been smarter. Intelligently cache textures for display in the viewport. Rapidly build and edit materials with the new Shader Tree enhancements. Spend less time rendering with the detail enhanced Intel Denoiser.

Workflow

Customization, simplification and I/O are central workflow concerns. Streamline tool usage with Conditional Commands, Get quick measurements through the Ruler/Grid MeshOp and reliably exchange assets with Rhino 7 native 3dm support.

Animation and Rigging

The new Wrap Effector offers a single, flexible entry point for both Wrap and Lattice style deformations. The action exporter makes exporting multiple animation clips for use in game engines a breeze.

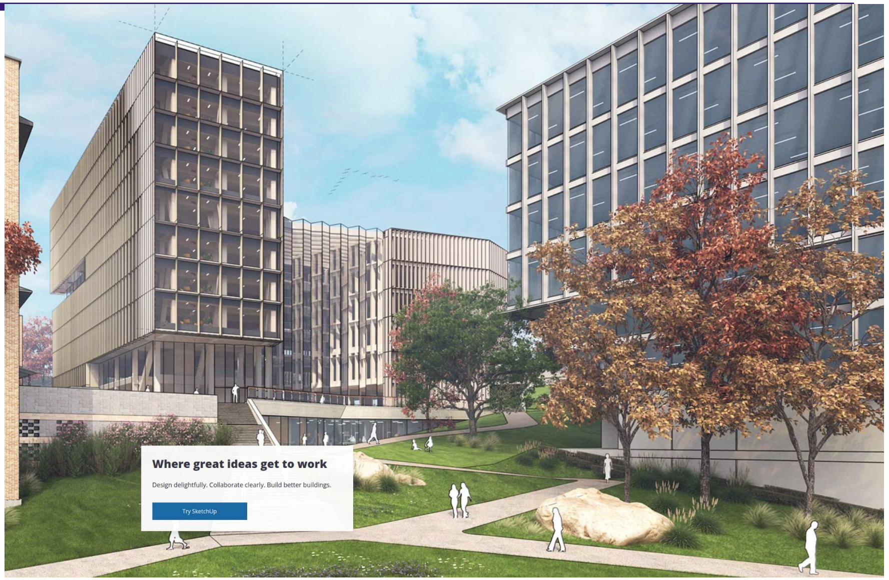
3D Modeling Software - Rhino [website]

- Rhino can create, edit, analyze, document, render, animate, and translate NURBS curves, surfaces and solids, subdivision geometry (SubD), point clouds, and polygon meshes. There are no limits on complexity, degree, or size beyond those of your hardware.

Rhino 7
Free-form, precisely.
Available now with SubD, Rhino.Inside.Revit, QuadRemesh, and more.
Learn more, Try, or Buy

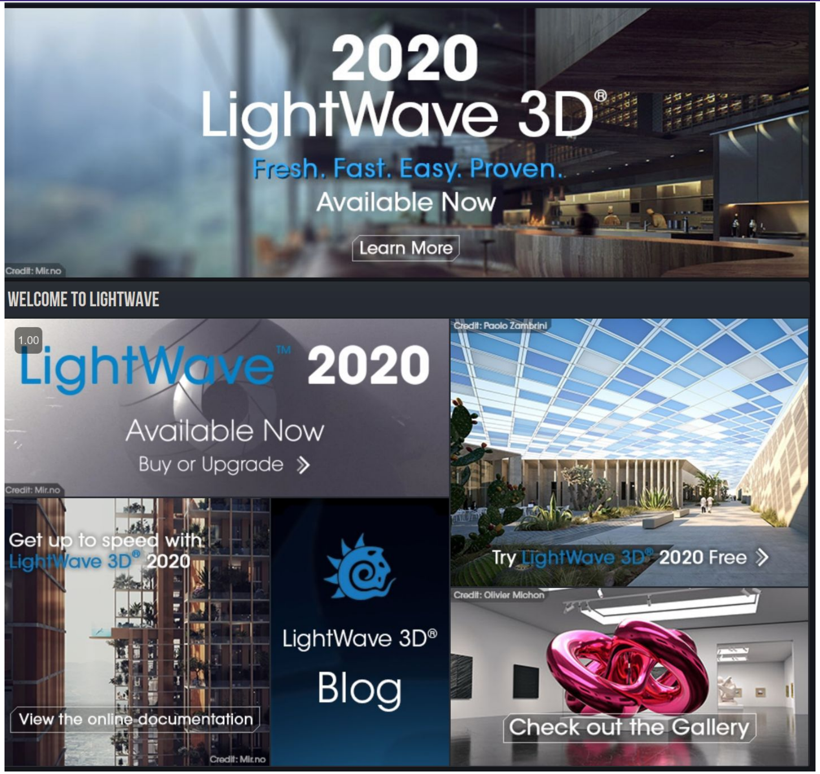
3D Modeling Software - Sketchup [website]

- *Design delightfully. Collaborate clearly. Build better buildings.*
- Free version available


Where great ideas get to work
Design delightfully. Collaborate clearly. Build better buildings.
Try SketchUp

3D Modeling Software - Lightwave [website]

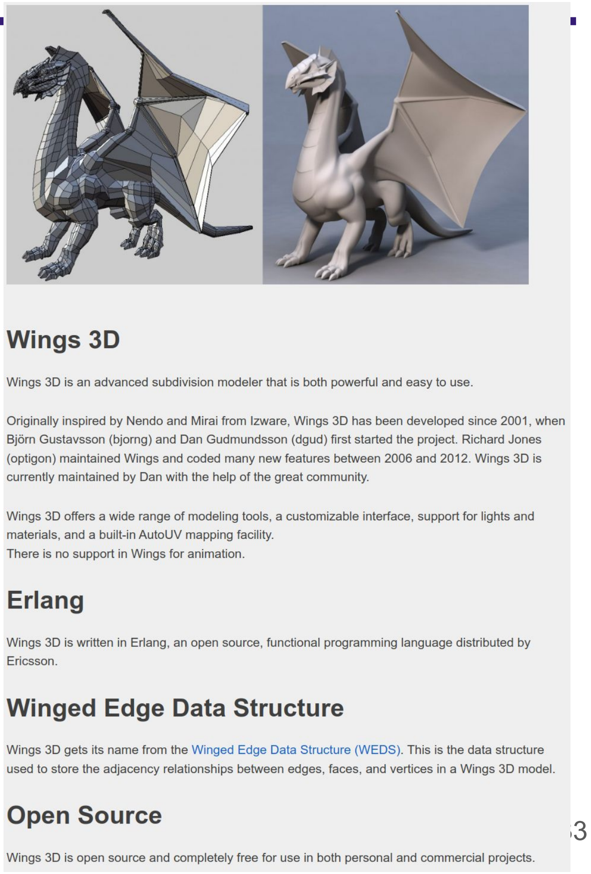
- Unclear how active they are now -- but they were a big name for a long time when I got into 3D graphics
  a. Larger presence in VFX for a long time



3D Modeling Software - Wings3D [website]

- *Wings 3D is an advanced subdivision modeler that is both powerful and easy to use.*
- *Originally inspired by Nendo and Mirai from Izware, Wings 3D has been developed since 2001, when Björn Gustavsson (bjorng) and Dan Gudmundsson (dgud) first started the project. Richard Jones (optigon) maintained Wings and coded many new features between 2006 and 2012. Wings 3D is currently maintained by Dan with the help of the great community.*
- *Wings 3D offers a wide range of modeling tools, a customizable interface, support for lights and materials, and a built-in AutoUV mapping facility.*
  There is no support in Wings for animation.

- The 3D modeling tool I was for a long time the most proficient in!



Wings 3D

Wings 3D is an advanced subdivision modeler that is both powerful and easy to use.

Originally inspired by Nendo and Mirai from Izware, Wings 3D has been developed since 2001, when Björn Gustavsson (bjorng) and Dan Gudmundsson (dgud) first started the project. Richard Jones (optigon) maintained Wings and coded many new features between 2006 and 2012. Wings 3D is currently maintained by Dan with the help of the great community.

Wings 3D offers a wide range of modeling tools, a customizable interface, support for lights and materials, and a built-in AutoUV mapping facility.
There is no support in Wings for animation.

Erlang

Wings 3D is written in Erlang, an open source, functional programming language distributed by Ericsson.

Winged Edge Data Structure

Wings 3D gets its name from the Winged Edge Data Structure (WEDS). This is the data structure used to store the adjacency relationships between edges, faces, and vertices in a Wings 3D model.

Open Source

Wings 3D is open source and completely free for use in both personal and commercial projects.

3D Modeling Software - Curvy3D [website]

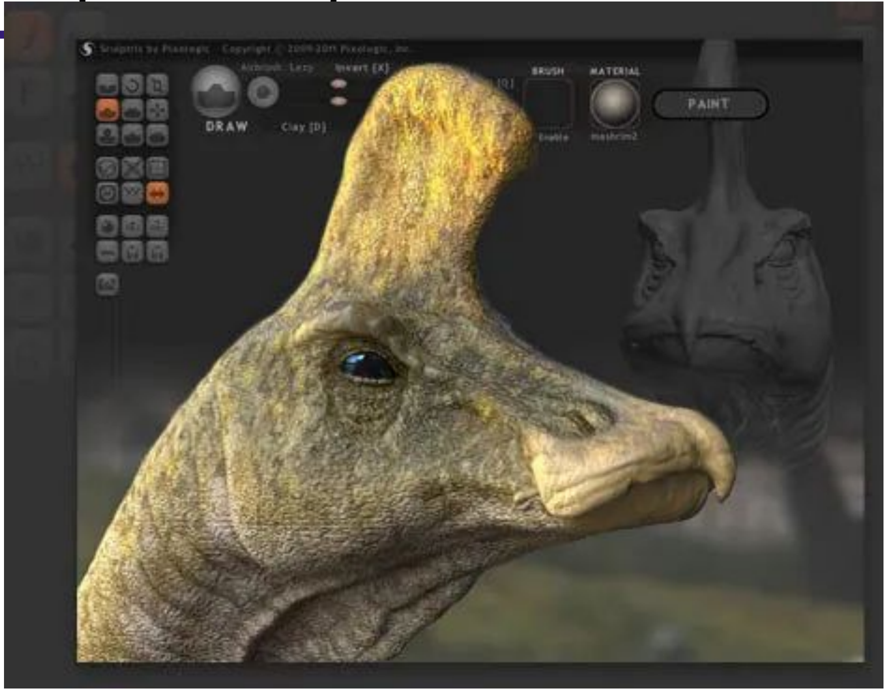
- *FUN 3D SCULPTING*
- *Use your drawing skills to model in 3D*
- *Draw 3D forms with your tablet or mouse.*
- *Blend forms together to make complex shapes.*
- *Detail the model using Curvy's surface sculpt tools.*
- *Paint bumps and colours onto the surface in 3D.*

More unique style of generating 3D models by drawing 2D curves/splines

Sculpted Head and Hat
Drawn in Curvy 3D - No Mesh Editing Required
Some examples of Curvy's Unique Sketched Curve Modelling Method
Cowboy
Sculpted in Curvy 3D 4.0
www.curvy3d.com

3D Modeling Software - Sculptris [website]

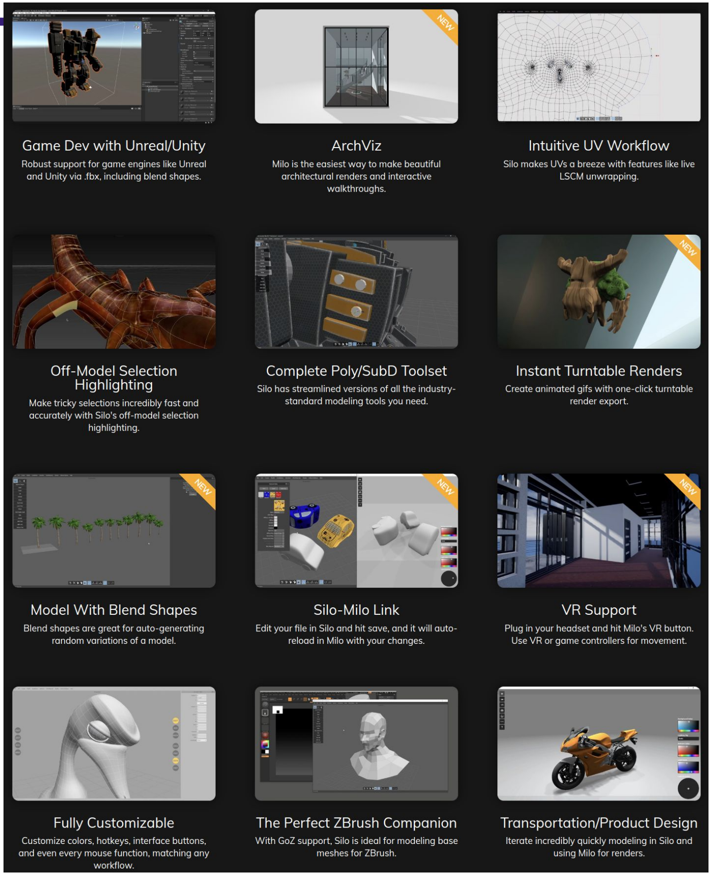
- *Sculptris is a free 3D modeling software, which exploits the modeling clay concept. It is unique in that anyone can pick it up and try it out without any technical knowledge.*
- My understanding is this is the free alternative to ZBrush
    a. I believe some of the same team worked on it.



3D Modeling Software - Silo [website]

- *Silo is a lightweight and lightning-fast 3D polygonal modeler and UV mapper, offering a deep, industry-grade toolset and easy-to-master workflow at an insanely affordable price. Its lightweight, focused, and portable nature have made it THE go-to pure modeler for over a decade, either used standalone or as the perfect modeling upgrade for all-in-one tools like Maya or Blender. It's ideal for creating base meshes for Mudbox or ZBrush (including GoZ support), and works seamlessly with Unity and Unreal game engines for creating all kinds of 3D game assets. Read about all the latest updates on our developer blog here).*

- Pure 3D modeling/texturing application for a reasonable price.



Game Dev with Unreal/Unity
Robust support for game engines like Unreal and Unity via .fbx, including blend shapes.

ArchViz
Milo is the easiest way to make beautiful architectural renders and interactive walkthroughs.

Intuitive UV Workflow
Silo makes UVs a breeze with features like live LSCM unwrapping.

Off-Model Selection Highlighting
Make tricky selections incredibly fast and accurately with Silo's off-model selection highlighting.

Complete Poly/SubD Toolset
Silo has streamlined versions of all the industry-standard modeling tools you need.

Instant Turntable Renders
Create animated gifs with one-click turntable render export.

Model With Blend Shapes
Blend shapes are great for auto-generating random variations of a model.

Silo-Milo Link
Edit your file in Silo and hit save, and it will auto-reload in Milo with your changes.

VR Support
Plug in your headset and hit Milo's VR button. Use VR or game controllers for movement.

Fully Customizable
Customize colors, hotkeys, interface buttons, and even every mouse function, matching any workflow.

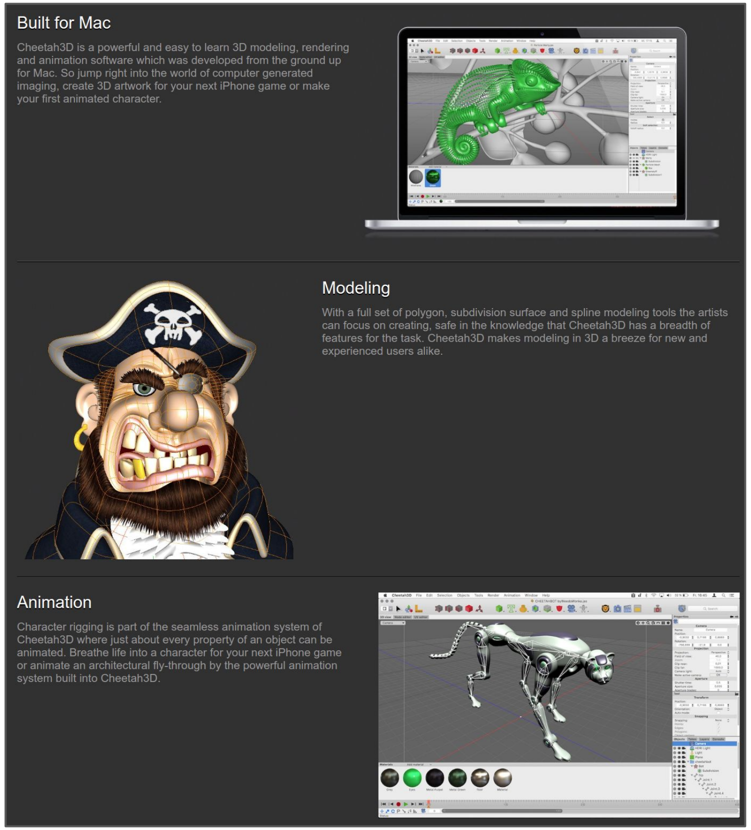
The Perfect ZBrush Companion
With GoZ support, Silo is ideal for modeling base meshes for ZBrush.

Transportation/Product Design
Iterate incredibly quickly modeling in Silo and using Milo for renders.

3D Modeling Software - Cheetah3d [website]

- *The powerful 3D modeling, rendering and animation software for [Built for Mac]*
- *Cheetah3D is a powerful and easy to learn 3D modeling, rendering and animation software which was developed from the ground up for Mac. So jump right into the world of computer generated imaging, create 3D artwork for your next iPhone game or make your first animated character.*

Mac specific 3D modeling tool



Built for Mac

Cheetah3D is a powerful and easy to learn 3D modeling, rendering and animation software which was developed from the ground up for Mac. So jump right into the world of computer generated imaging, create 3D artwork for your next iPhone game or make your first animated character.

Modeling

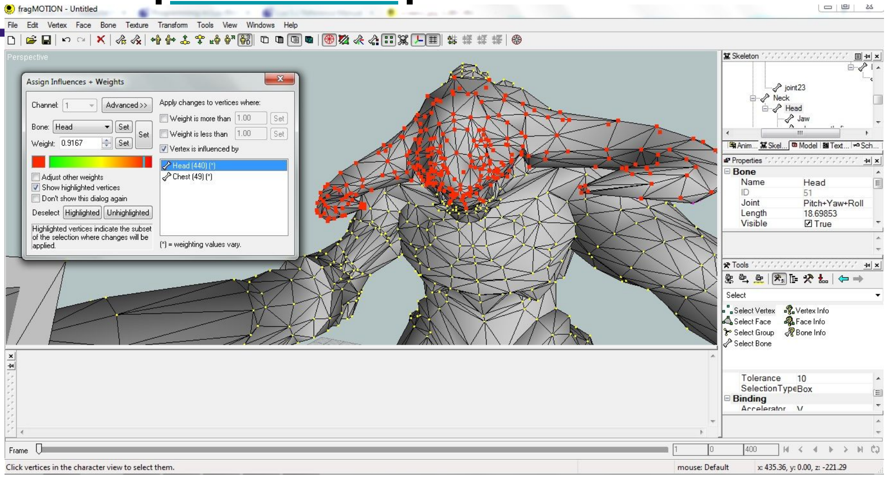
With a full set of polygon, subdivision surface and spline modeling tools the artists can focus on creating, safe in the knowledge that Cheetah3D has a breadth of features for the task. Cheetah3D makes modeling in 3D a breeze for new and experienced users alike.

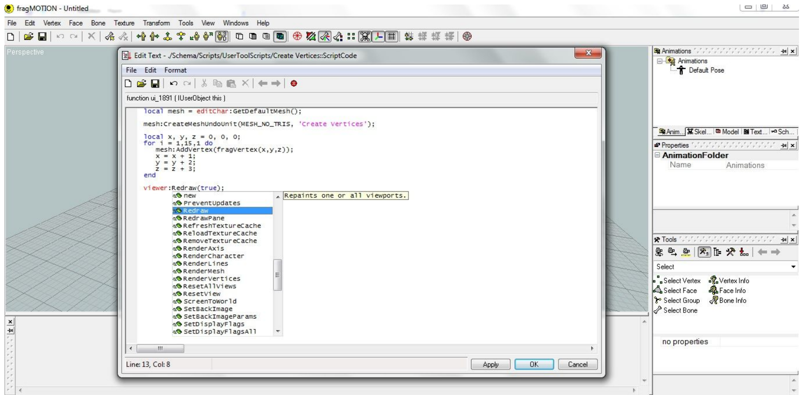
Animation

Character rigging is part of the seamless animation system of Cheetah3D where just about every property of an object can be animated. Breathe life into a character for your next iPhone game or animate an architectural fly-through by the powerful animation system built into Cheetah3D.

3D Modeling Software - FragMotion [website]

- ***fragMOTION** is a 3D modeler used for the creation and animation of characters.*
- Another early 3D modeling tool targeted more for indie game developers





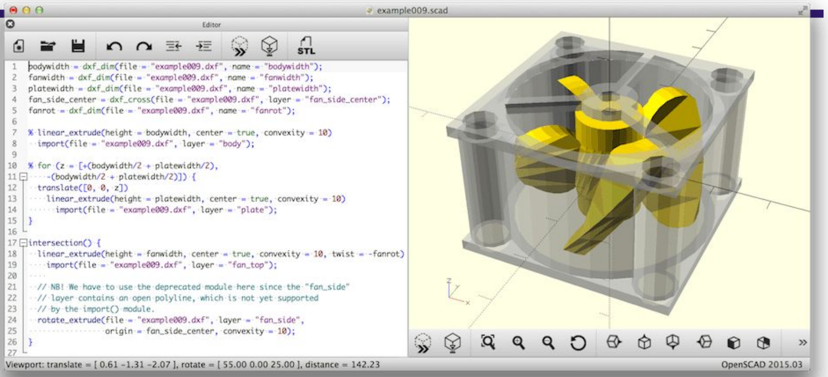
```
function ui_1891 ( IUserObject this )
local mesh = editChar:GetDefaultMesh();

mesh:CreateMeshUndoUnit(MESH_NO_TRIS, 'Create Vertices');

local x, y, z = 0, 0, 0;
for i = 1,15,1 do
   mesh:AddVertex(fragVertex(x,y,z));
   x = x + 1;
   y = y + 2;
   z = z + 3;
end

viewer:Redraw(true);
```

```
new
PreventUpdates
Redraw
RedrawPane
RefreshTextureCache
ReloadTextureCache
RemoveTextureCache
RenderAxis
RenderCharacter
RenderLines
RenderMesh
RenderVertices
ResetAllViews
ResetView
ScreenToWorld
SetBackImage
SetBackImageParams
SetDisplayFlags
SetDisplayFlagsAll
```

Repaints one or all viewports.

3D Modeling Software - OpenSCAD [website]

- *OpenSCAD is software for creating solid 3D CAD models. It is free software and available for Linux/UNIX, Windows and Mac OS X. Unlike most free software for creating 3D models (such as Blender) it does not focus on the artistic aspects of 3D modelling but instead on the CAD aspects. Thus it might be the application you are looking for when you are planning to create 3D models of machine parts but pretty sure is not what you are looking for when you are more interested in creating computer-animated movies.*
- *OpenSCAD is not an interactive modeller. Instead it is something like a 3D-compiler that reads in a script file that describes the object and renders the 3D model from this script file. This gives you (the designer) full control over the modelling process and enables you to easily change any step in the modelling process or make designs that are defined by configurable parameters.*
- *OpenSCAD provides two main modelling techniques: First there is constructive solid geometry (aka CSG) and second there is extrusion of 2D outlines. Autocad DXF files can be used as the data exchange format for such 2D outlines. In addition to 2D paths for extrusion it is also possible to read design parameters from DXF files. Besides DXF files OpenSCAD can read and create 3D models in the STL and OFF file formats.*



```
bodywidth = dxf_dim(file = "example009.dxf", name = "bodywidth");
fanwidth = dxf_dim(file = "example009.dxf", name = "fanwidth");
platewidth = dxf_dim(file = "example009.dxf", name = "platewidth");
fan_side_center = dxf_cross(file = "example009.dxf", layer = "fan_side_center");
fanrot = dxf_dim(file = "example009.dxf", name = "fanrot");

% linear_extrude(height = bodywidth, center = true, convexity = 10)
  import(file = "example009.dxf", layer = "body");

% for (z = [+(bodywidth/2 + platewidth/2),
    -(bodywidth/2 + platewidth/2)]) {
  translate([0, 0, z])
    linear_extrude(height = platewidth, center = true, convexity = 10)
      import(file = "example009.dxf", layer = "plate");
}

intersection() {
  linear_extrude(height = fanwidth, center = true, convexity = 10, twist = -fanrot)
    import(file = "example009.dxf", layer = "fan_top");

  // NB! We have to use the deprecated module here since the "fan_side"
  // layer contains an open polyline, which is not yet supported
  // by the import() module.
  rotate_extrude(file = "example009.dxf", layer = "fan_side",
                 origin = fan_side_center, convexity = 10);
}
```

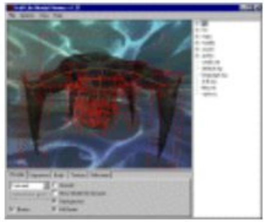
Viewport: translate = [0.61 -1.31 -2.07], rotate = [55.00 0.00 25.00], distance = 142.23

3D Modeling Software - Milkshape3D [website]

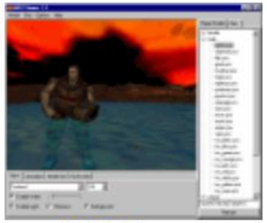
- MilkShape 3D is a low-polygon modeler, which was initially designed for Half-Life. By and by many file formats and features have been added.
- MilkShape 3D has all basic operations like select, move, rotate, scale, extrude, turn edge, subdivide, just to mention a few. MilkShape 3D also allows low-level editing with the vertex and face tool. Standard and extended primitives like spheres, boxes, cylinders, etc. are available too.
- MilkShape 3D is a skeletal animator. This allows to export to morph target animation like the ones in the Quake model formats or to export to skeletal animations like Half-Life, Genesis3d, Unreal, etc.
- MilkShape 3D currently supports 70 different file formats! Download MilkShape 3D yourself and see how easy it is to create models!

- More for nostalgic purposes

MilkShape 3D
Create and edit models for games like Quake I, II, III, Half-Life, Unreal Tournament, ncluding full animation! Or write your custom export plugin with the SDK!



HL Model Viewer
View models and change skins for HL, TFC and CS!

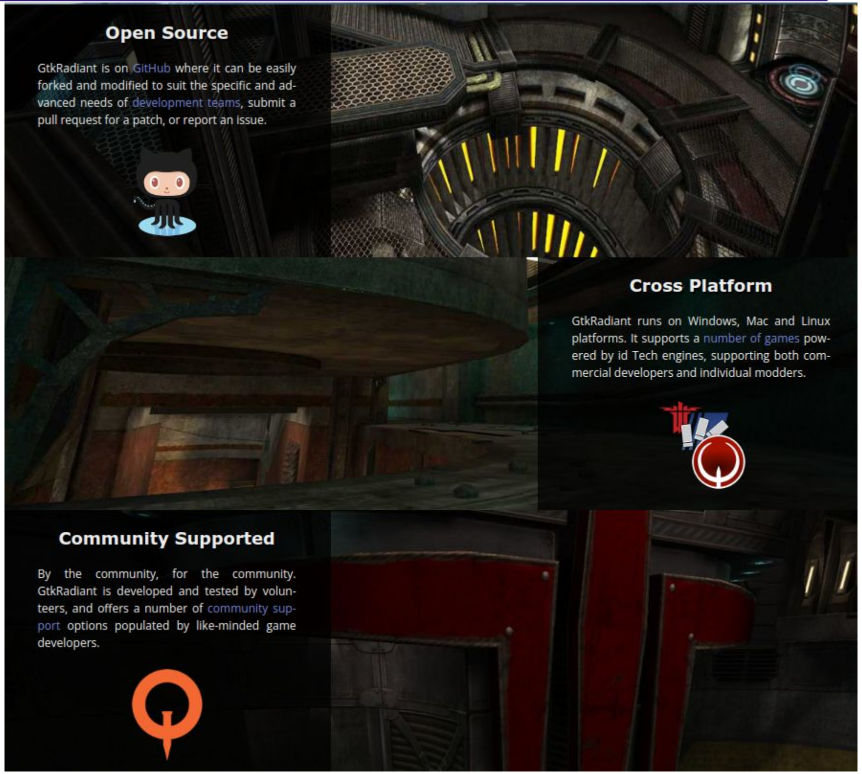


MD2 Viewer
View Quake2 models directly from PAK files!

3D Modeling Software - gtkRadiant [website]

- *GtkRadiant is the official level design toolchain for games powered by id Tech engines from id Software, and is maintained by a community of volunteers. GtkRadiant is powered by the GTK+ Project and released under a GPL license.*

Neat open source, longstanding project primarily for indoor level creation.



Open Source

GtkRadiant is on GitHub where it can be easily forked and modified to suit the specific and advanced needs of development teams, submit a pull request for a patch, or report an issue.

Cross Platform

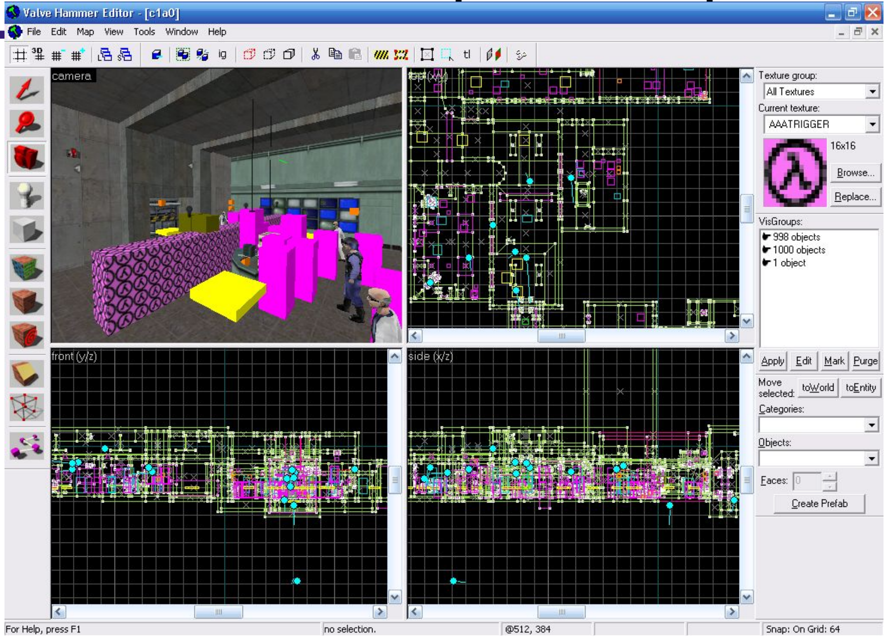
GtkRadiant runs on Windows, Mac and Linux platforms. It supports a number of games powered by id Tech engines, supporting both commercial developers and individual modders.

Community Supported

By the community, for the community. GtkRadiant is developed and tested by volunteers, and offers a number of community support options populated by like-minded game developers.

3D Modeling Software - Valve Hammer Editor [website]

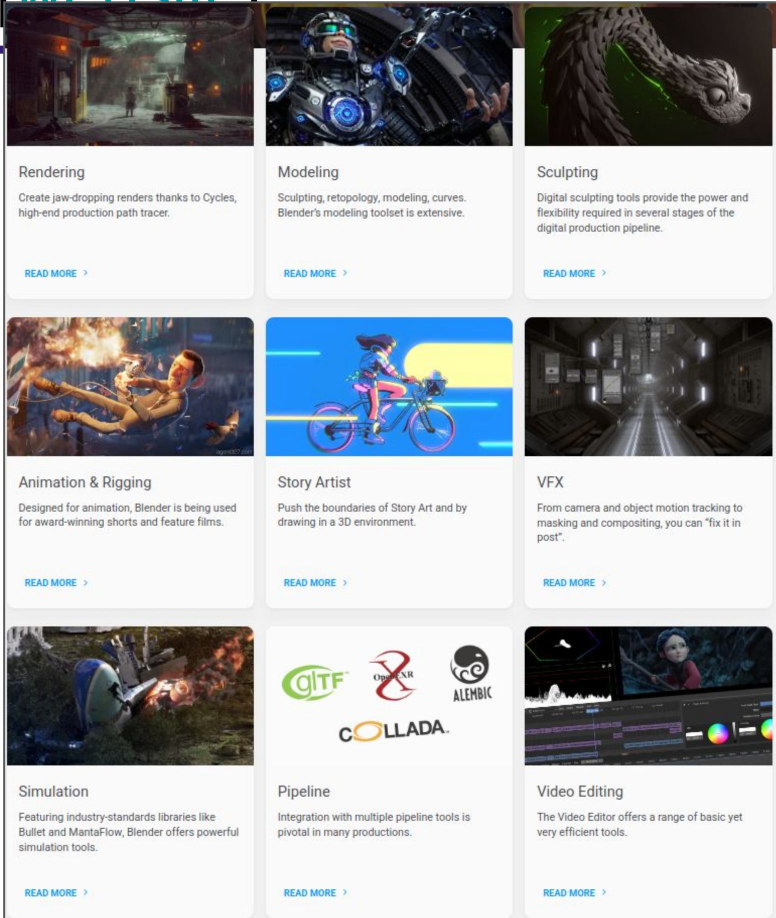
- *Valve Hammer Editor (more informally known as Hammer, and previously called Worldcraft) is the official mapping tool for Valve's engines: GoldSrc, Source, and Source 2. It is also included in every game made with Source Engine that is not a mod or third-party.*
- Primarily a Constructive Solid Geometry (CSG) tool for modeling Indoor environments.



3D Modeling Software - Blender3D [website]

- *Mission*
- *Get the world's best 3D CG technology in the hands of artists as free/open source software.*
- *Vision*
- *Everyone should be free to create 3D CG content, with free technical and creative production means and free access to markets.*

One of the most popular and powerful free and open source 3D modelers. Often integrated in many commercial workflows.



Rendering
Create jaw-dropping renders thanks to Cycles, high-end production path tracer.
READ MORE ›

Modeling
Sculpting, retopology, modeling, curves. Blender's modeling toolset is extensive.
READ MORE ›

Sculpting
Digital sculpting tools provide the power and flexibility required in several stages of the digital production pipeline.
READ MORE ›

Animation & Rigging
Designed for animation, Blender is being used for award-winning shorts and feature films.
READ MORE ›

Story Artist
Push the boundaries of Story Art and by drawing in a 3D environment.
READ MORE ›

VFX
From camera and object motion tracking to masking and compositing, you can "fix it in post".
READ MORE ›

Simulation
Featuring industry-standards libraries like Bullet and MantaFlow, Blender offers powerful simulation tools.
READ MORE ›

Pipeline
Integration with multiple pipeline tools is pivotal in many productions.
READ MORE ›

Video Editing
The Video Editor offers a range of basic yet very efficient tools.
READ MORE ›

Blender3D

Blender3D

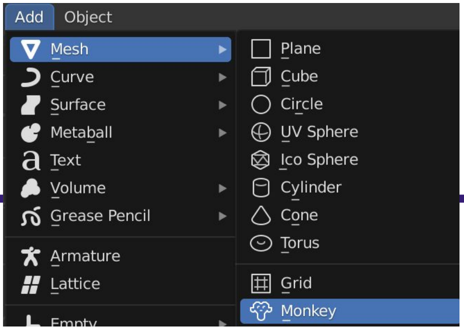
- Blender3D is a full suite of 3D (and 2D) graphics tools used in the game, VFX, and motion picture industry.
    - Commercial movies, games, and television shows have been produced using Blender3D

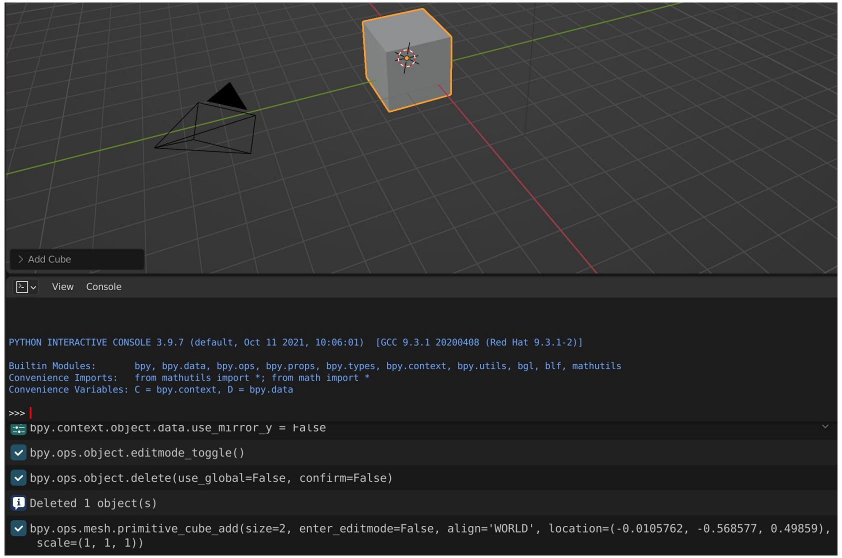
Blender3D is free -- and is getting more powerful every year!

Programming Blender 3D

Origin of this Talk

- The idea of this talk was born out of a computational geometry course that I teach
  - Within that course we implement several geometry algorithms in C++ and libsdl library in two-dimensions
- In order to start implementing in 3D however, I could not assume students knew OpenGL/Vulkan/Metal/D3D
  - So what better tool than Blender 3D which had many mesh operations and an easy scripting interface to access them.
  - Teaching students a concrete skill (i.e. Blender 3D) is also a win for me!

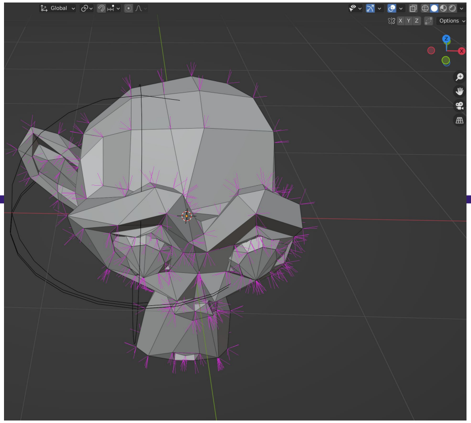


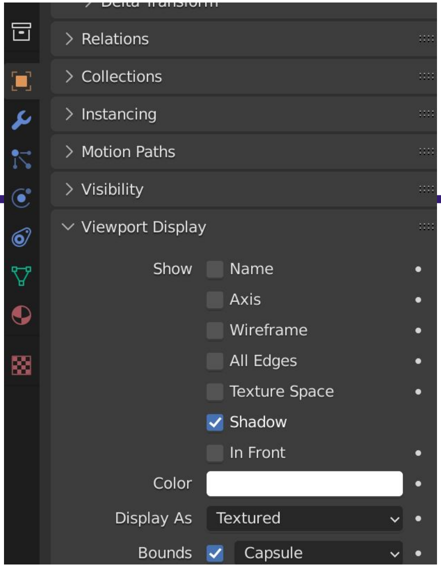


```
PYTHON INTERACTIVE CONSOLE 3.9.7 (default, Oct 11 2021, 10:06:01)  [GCC 9.3.1 20200408 (Red Hat 9.3.1-2)]

Builtin Modules:       bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, bpy.utils, bgl, blf, mathutils
Convenience Imports:   from mathutils import *; from math import *
Convenience Variables: C = bpy.context, D = bpy.data

>>> 
bpy.context.object.data.use_mirror_y = False
bpy.ops.object.editmode_toggle()
bpy.ops.object.delete(use_global=False, confirm=False)
Deleted 1 object(s)
bpy.ops.mesh.primitive_cube_add(size=2, enter_editmode=False, align='WORLD', location=(-0.0105762, -0.568577, 0.49859),
 scale=(1, 1, 1))
```

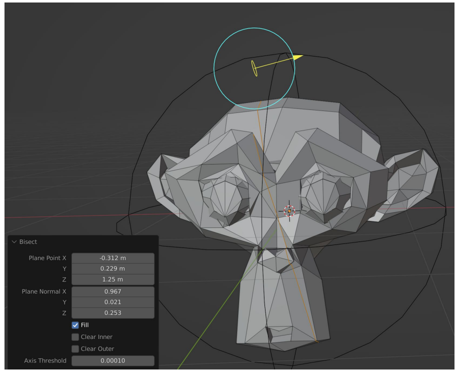
Brainstorming (1/2)

- So I thought of several ideas of how to get students started:
  - Computing Normals
  - Bisection
  - Convex Hull
  - Bounding Boxes
- I settled on bounding boxes, as it touches on enough interesting ideas for programming in Blender 3D
  - The rest remained candidates for incorporating into a final project!
  - (And homework for you now!)

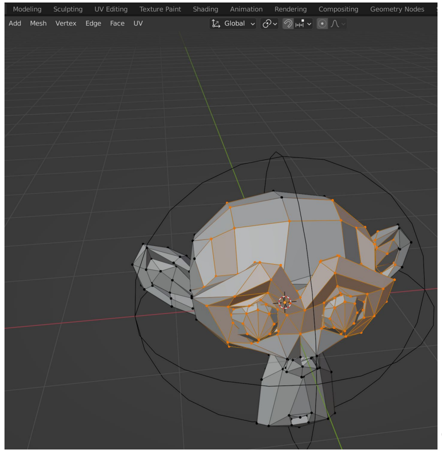



Relations
Collections
Instancing
Motion Paths
Visibility
Viewport Display

Show	Name
	Axis
	Wireframe
	All Edges
	Texture Space
	✓ Shadow
	In Front
Color	
Display As	Textured
Bounds	✓ Capsule


Bisect

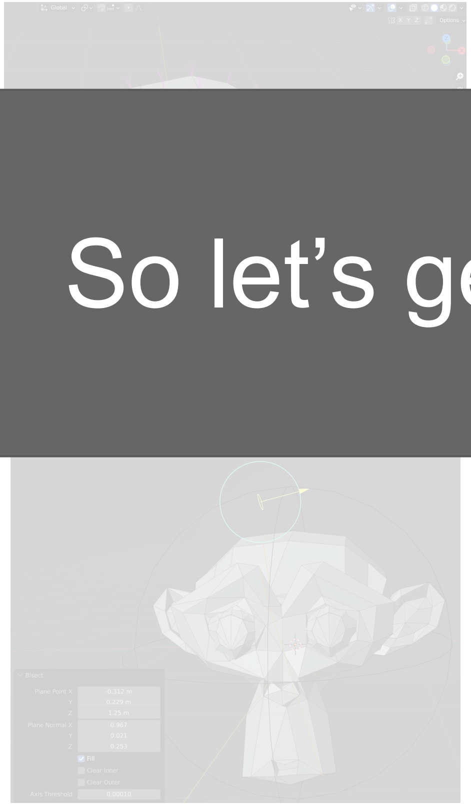
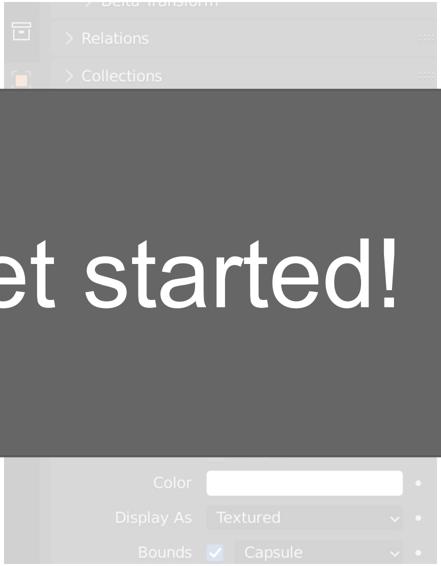
Plane Point X	-0.312 m
Y	0.229 m
Z	1.25 m
Plane Normal X	0.967
Y	0.021
Z	0.253
	✓ Fill
	Clear Inner
	Clear Outer
Axis Threshold	0.00010

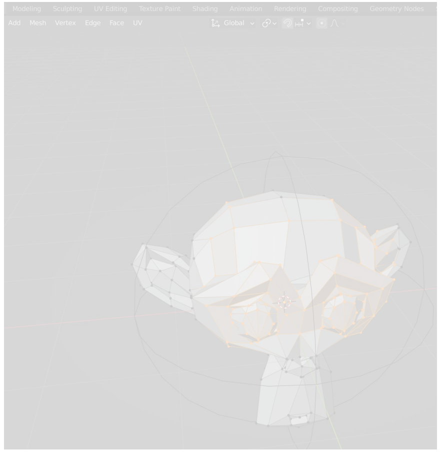


Brainstorming (2/2)

- So I thought of several ideas of how to get students started:
  - Computing Normals
  - Bisection
  - Convex Hull
  - **Bounding Boxes**
- I settled on bounding boxes, as it touches on enough interesting ideas for programming in Blender3D
  - The rest remained candidates for incorporating into a final project!
  - (And homework for you now!)

So let's get started!

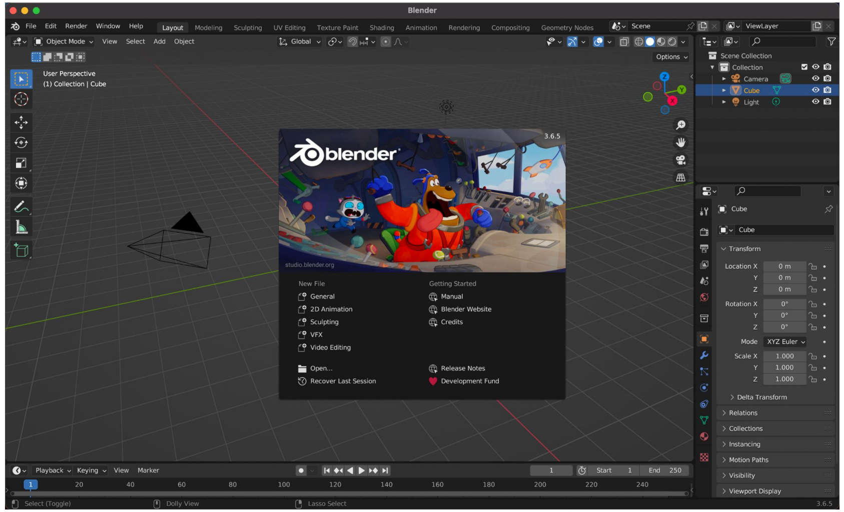





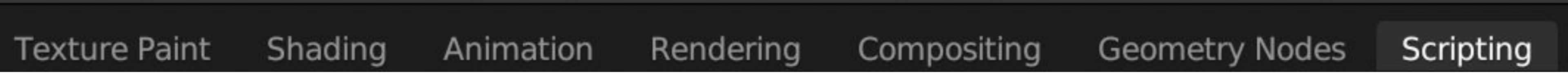
Writing Python Scripts in Blender 3D

Install Blender 3D

- I'll assume you have installed Blender 3D
  - I'll write scripts using Blender 3.6.5, but these scripts should largely be compatible with most every version of Blender 3.x.x
    - Nothing too fancy going on today
- The last assumption I'll make is that you have used Blender 3D at least a little bit
  - Minimum Requirements: You can navigate with the mouse, extrude some faces on a cube, and have spent a few hours in the program
  - But that's about it! That's great news if you're a programmer building plugins to support a project, and great news if you're already an expert artist!

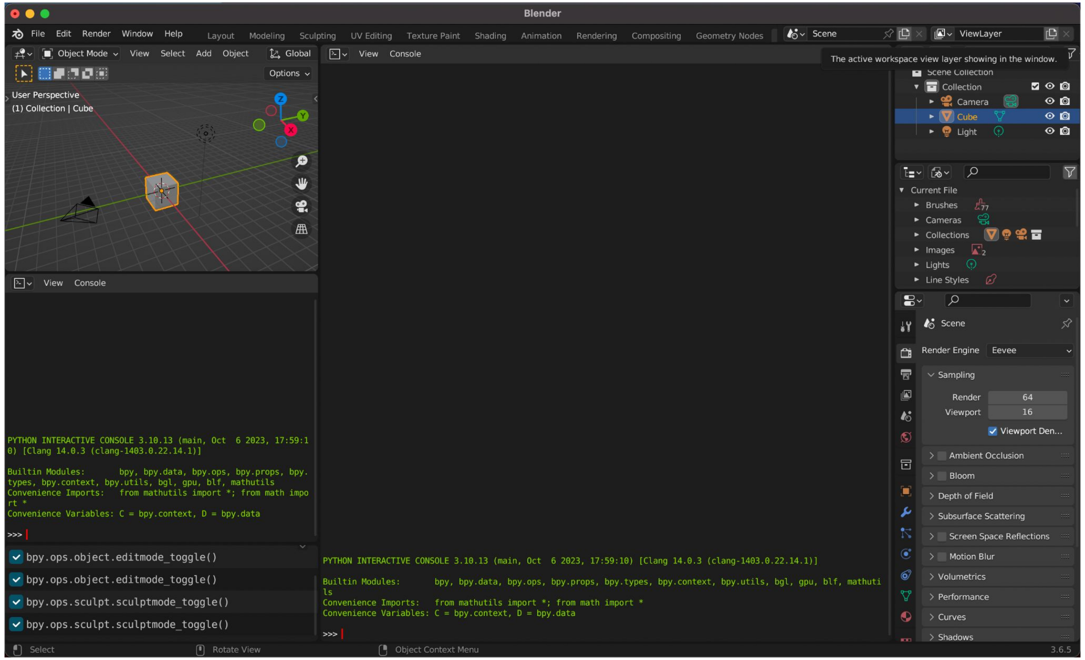


https://www.blender.org/download/



Scripting Layout (1/5)

- We are primarily going to be working from the **Scripting Workspace**
  - Here’s what the scripting layout looks like



Render Engine: Eevee

Render: 64

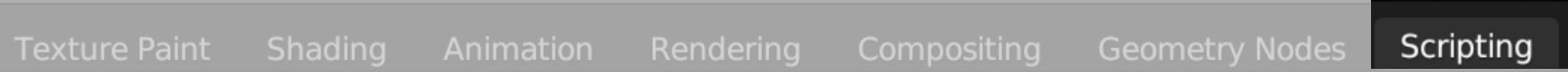
Viewport: 16

✓ Viewport Den...

```
PYTHON INTERACTIVE CONSOLE 3.10.13 (main, Oct  6 2023, 17:59:1
0) [Clang 14.0.3 (clang-1403.0.22.14.1)]

Builtin Modules:       bpy, bpy.data, bpy.ops, bpy.props, bpy.
types, bpy.context, bpy.utils, bgl, gpu, blf, mathutils
Convenience Imports:   from mathutils import *; from math impo
rt *
Convenience Variables: C = bpy.context, D = bpy.data

>>>
```

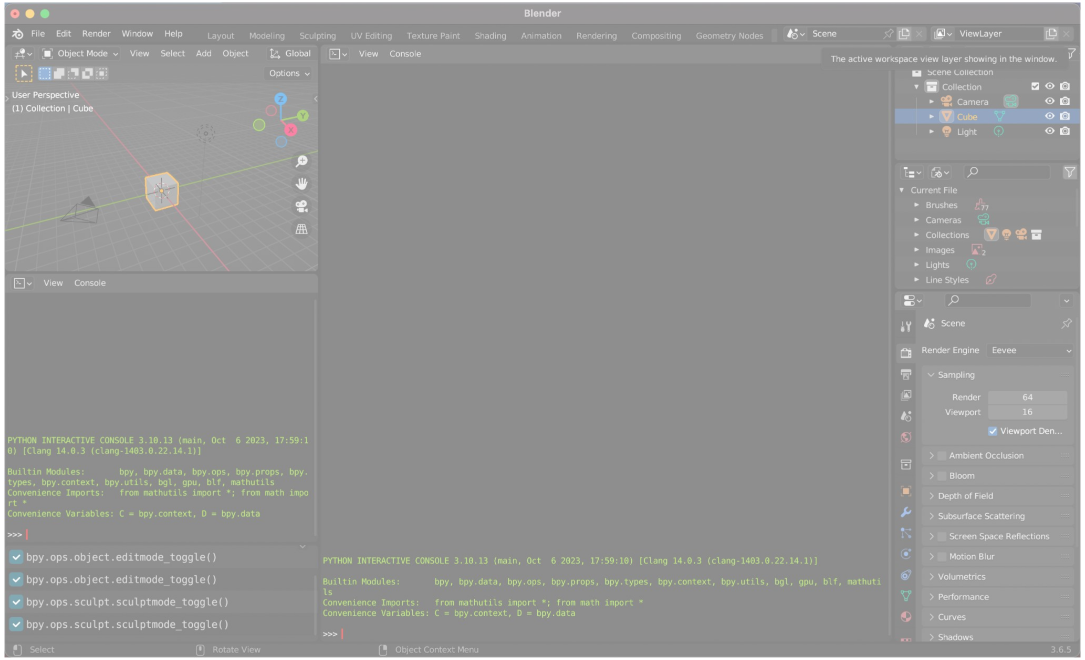
```
✓ bpy.ops.object.editmode_toggle()
✓ bpy.ops.object.editmode_toggle()
✓ bpy.ops.sculpt.sculptmode_toggle()
✓ bpy.ops.sculpt.sculptmode_toggle()
```

```
PYTHON INTERACTIVE CONSOLE 3.10.13 (main, Oct  6 2023, 17:59:10) [Clang 14.0.3 (clang-1403.0.22.14.1)]

Builtin Modules:       bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, bpy.utils, bgl, gpu, blf, mathuti
ls
Convenience Imports:   from mathutils import *; from math import *
Convenience Variables: C = bpy.context, D = bpy.data

>>>
```



Scripting Layout (2/5)

- **Script Workspace**
  - For scripting!
- Python Console
  - Useful for typing in commands, querying information, and getting fast feedback
- Info Log
  - Tells you the results of operations occurring in the viewport (e.g. moving around)
- Text Editor
  - Used for executing larger scripts

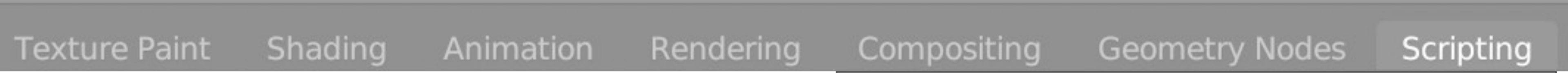


Render Engine Eevee
Render 64
Viewport 16

```
PYTHON INTERACTIVE CONSOLE 3.10.13 (main, Oct  6 2023, 17:59:1
0) [Clang 14.0.3 (clang-1403.0.22.14.1)]

Builtin Modules:       bpy, bpy.data, bpy.ops, bpy.props, bpy.
types, bpy.context, bpy.utils, bgl, gpu, blf, mathutils
Convenience Imports:   from mathutils import *; from math impo
rt *
Convenience Variables: C = bpy.context, D = bpy.data

>>>
```

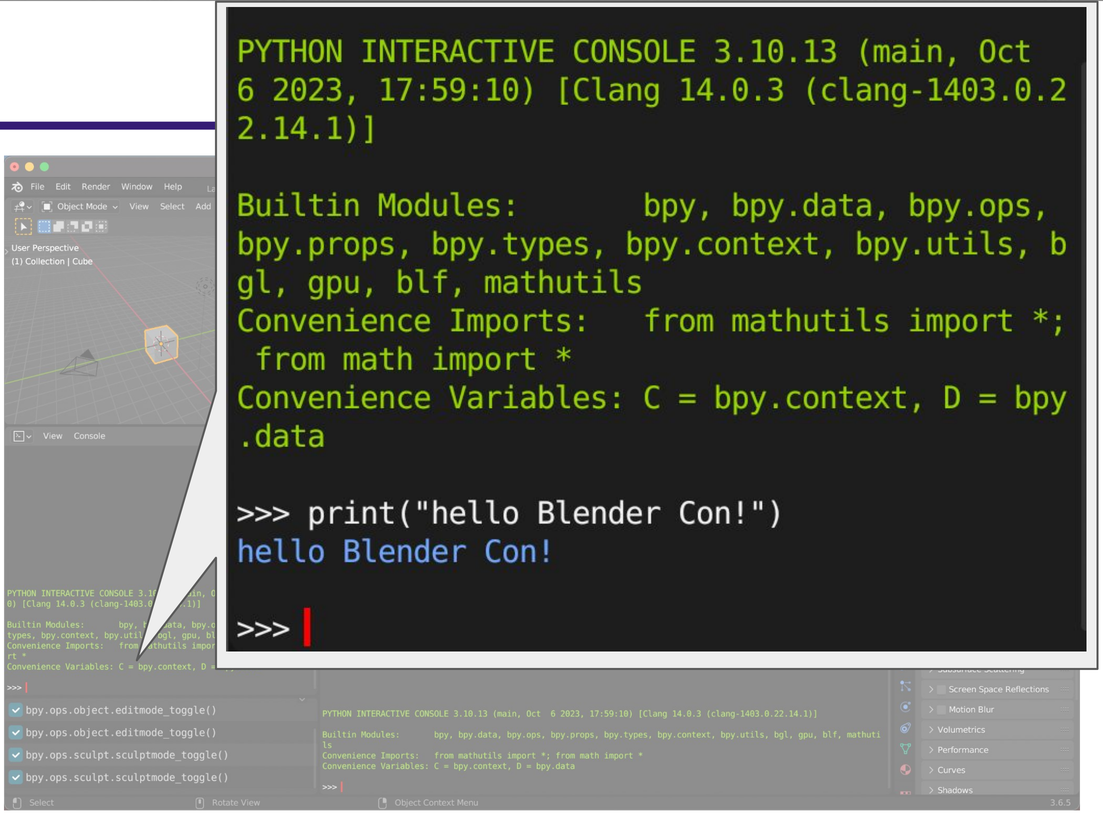
```
bpy.ops.object.editmode_toggle()
bpy.ops.object.editmode_toggle()
bpy.ops.sculpt.sculptmode_toggle()
bpy.ops.sculpt.sculptmode_toggle()
```

```
PYTHON INTERACTIVE CONSOLE 3.10.13 (main, Oct  6 2023, 17:59:10) [Clang 14.0.3 (clang-1403.0.22.14.1)]

Builtin Modules:       bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, bpy.utils, bgl, gpu, blf, mathuti
ls
Convenience Imports:   from mathutils import *; from math import *
Convenience Variables: C = bpy.context, D = bpy.data

>>>
```



Scripting Layout (3/5)

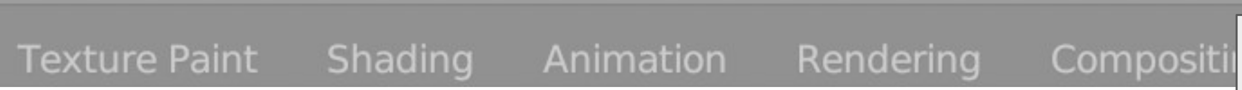
- Script Workspace
  - For scripting!
- **Python Console**
  - Useful for typing in commands, querying information, and getting fast feedback
- Info Log
  - Tells you the results of operations occurring in the viewport (e.g. moving around)
- Text Editor
  - Used for executing larger scripts



```
PYTHON INTERACTIVE CONSOLE 3.10.13 (main, Oct
6 2023, 17:59:10) [Clang 14.0.3 (clang-1403.0.2
2.14.1)]

Builtin Modules:       bpy, bpy.data, bpy.ops,
bpy.props, bpy.types, bpy.context, bpy.utils, b
gl, gpu, blf, mathutils
Convenience Imports:   from mathutils import *;
 from math import *
Convenience Variables: C = bpy.context, D = bpy
.data

>>> print("hello Blender Con!")
hello Blender Con!

>>> 
```

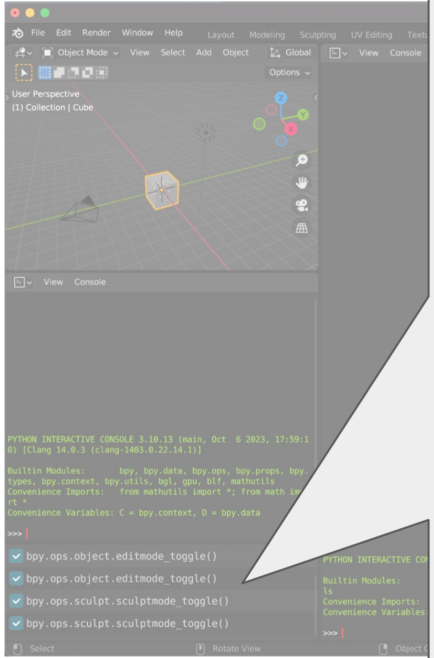
```
bpy.ops.object.editmode_toggle()
bpy.ops.object.editmode_toggle()
bpy.ops.sculpt.sculptmode_toggle()
bpy.ops.sculpt.sculptmode_toggle()
```

```
PYTHON INTERACTIVE CONSOLE 3.10.13 (main, Oct  6 2023, 17:59:10) [Clang 14.0.3 (clang-1403.0.22.14.1)]

Builtin Modules:       bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, bpy.utils, bgl, gpu, blf, mathuti
ls
Convenience Imports:   from mathutils import *; from math import *
Convenience Variables: C = bpy.context, D = bpy.data

>>> 
```



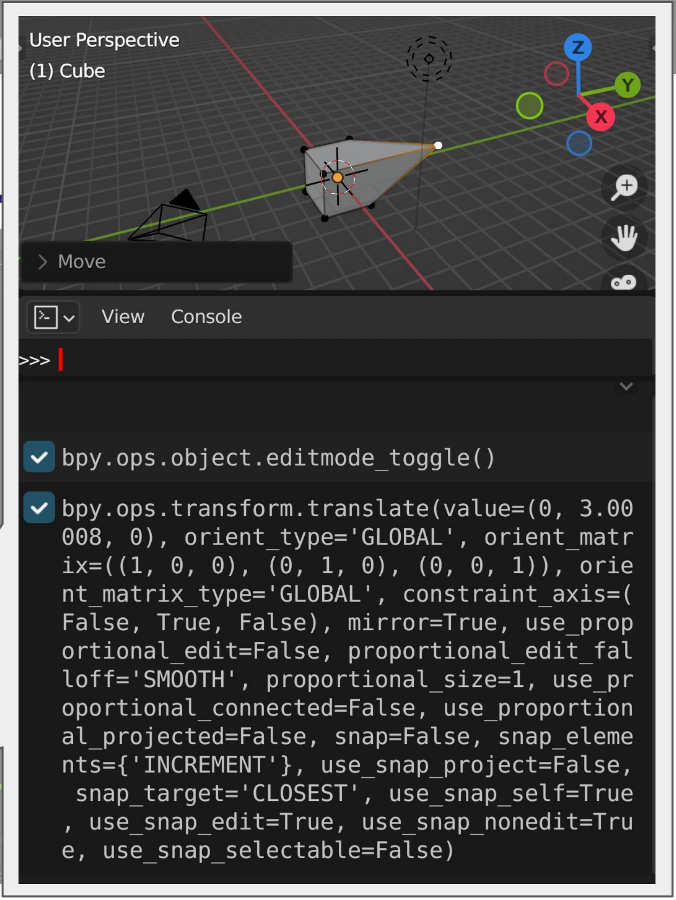
Scripting Layout (4/5)

- Script Workspace
  - For scripting!
- Python Console
  - Useful for typing in commands, querying information, and getting fast feedback
- **Info Log**
  - Tells you the results of operations occurring in the viewport (e.g. moving around)
- Text Editor
  - Used for executing larger scripts

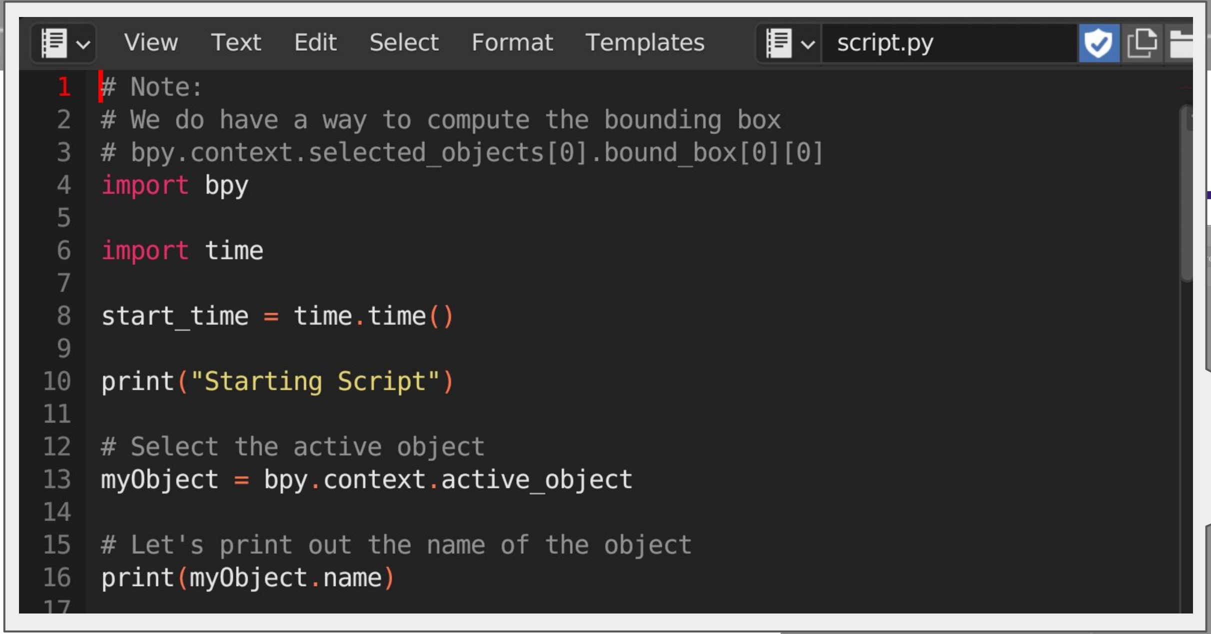


```
PYTHON INTERACTIVE CONSOLE 3.10.13 (main, Oct  6 2023, 17:59:1
0) [Clang 14.0.3 (clang-1403.0.22.14.1)]

Builtin Modules:       bpy, bpy.data, bpy.ops, bpy.props, bpy.
types, bpy.context, bpy.utils, bgl, gpu, blf, mathutils
Convenience Imports:   from mathutils import *; from math impo
rt *
Convenience Variables: C = bpy.context, D = bpy.data

>>>
```

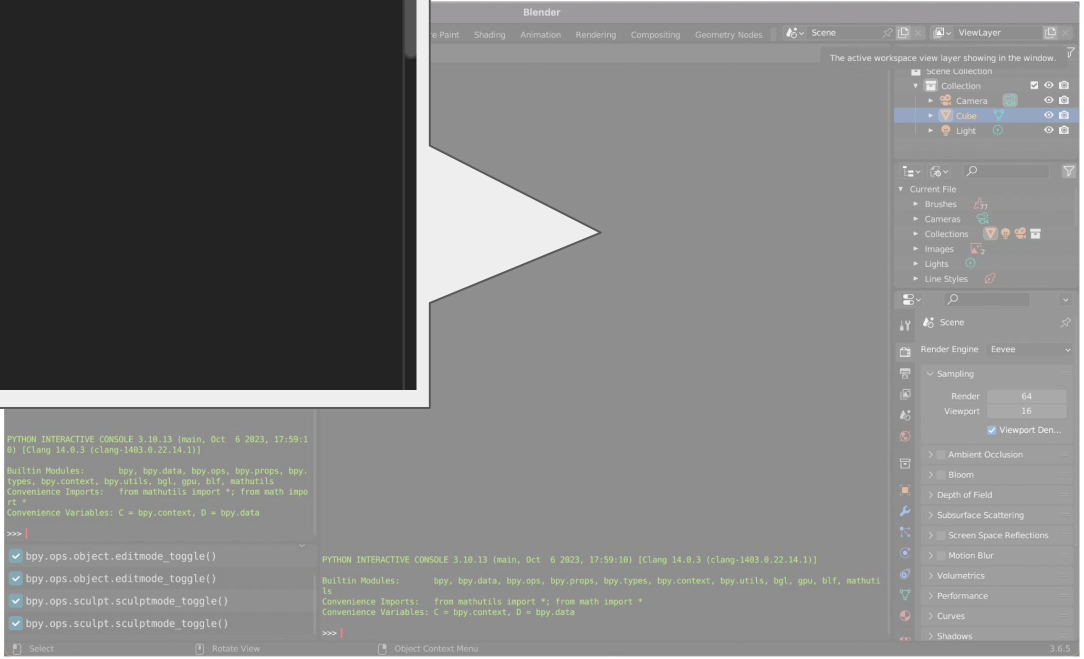
```
bpy.ops.object.editmode_toggle()
bpy.ops.object.editmode_toggle()
bpy.ops.sculpt.sculptmode_toggle()
bpy.ops.sculpt.sculptmode_toggle()
```



```
>>>

bpy.ops.object.editmode_toggle()
bpy.ops.transform.translate(value=(0, 3.00
008, 0), orient_type='GLOBAL', orient_matr
ix=((1, 0, 0), (0, 1, 0), (0, 0, 1)), orie
nt_matrix_type='GLOBAL', constraint_axis=(
False, True, False), mirror=True, use_prop
ortional_edit=False, proportional_edit_fal
loff='SMOOTH', proportional_size=1, use_pr
oportional_connected=False, use_proportion
al_projected=False, snap=False, snap_eleme
nts={'INCREMENT'}, use_snap_project=False,
 snap_target='CLOSEST', use_snap_self=True
, use_snap_edit=True, use_snap_nonedit=Tru
e, use_snap_selectable=False)
```



```python
# Note:
# We do have a way to compute the bounding box
# bpy.context.selected_objects[0].bound_box[0][0]
import bpy

import time

start_time = time.time()

print("Starting Script")

# Select the active object
myObject = bpy.context.active_object

# Let's print out the name of the object
print(myObject.name)
```



```
PYTHON INTERACTIVE CONSOLE 3.10.13 (main, Oct  6 2023, 17:59:1
0) [Clang 14.0.3 (clang-1403.0.22.14.1)]

Builtin Modules:       bpy, bpy.data, bpy.ops, bpy.props, bpy.
types, bpy.context, bpy.utils, bgl, gpu, blf, mathutils
Convenience Imports:   from mathutils import *; from math impo
rt *
Convenience Variables: C = bpy.context, D = bpy.data

>>>
```

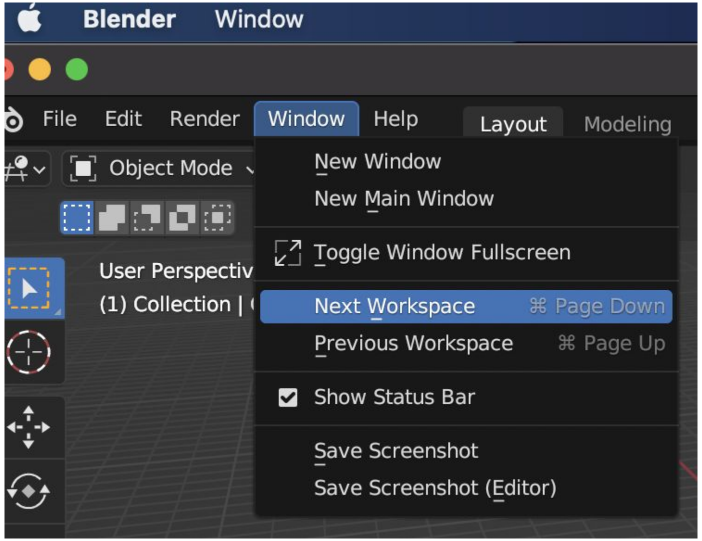
```
bpy.ops.object.editmode_toggle()
bpy.ops.object.editmode_toggle()
bpy.ops.sculpt.sculptmode_toggle()
bpy.ops.sculpt.sculptmode_toggle()
```

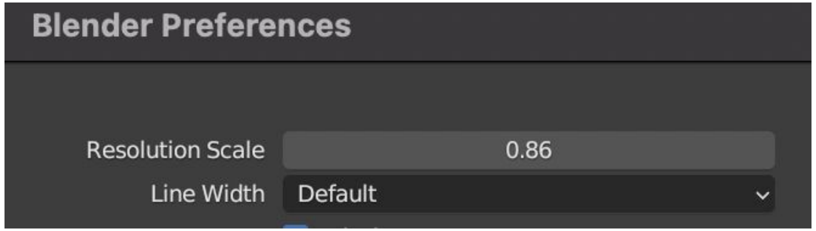
```
PYTHON INTERACTIVE CONSOLE 3.10.13 (main, Oct  6 2023, 17:59:10) [Clang 14.0.3 (clang-1403.0.22.14.1)]

Builtin Modules:       bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, bpy.utils, bgl, gpu, blf, mathuti
ls
Convenience Imports:   from mathutils import *; from math import *
Convenience Variables: C = bpy.context, D = bpy.data

>>>
```

- Info Log
  - Tells you the results of operations occurring in the viewport (e.g. moving around)
- **Text Editor**
  - Used for executing larger scripts

(Aside) If you cannot find the Scripting Workspace

- Scripting Workspace is usually the last tab.
  - Just scroll over the menu bar and scroll your mouse wheel
  - Or otherwise use 'page down'
- If you're on a Mac laptop with a small screen, you can navigate to the next workspace
  - Mac users without a page down button:
    - `Cmd + fn + down`
- Another option is to scale down your display a bit
  - Edit -> Preferences -> Adjust Resolution Scale




Blender Preferences

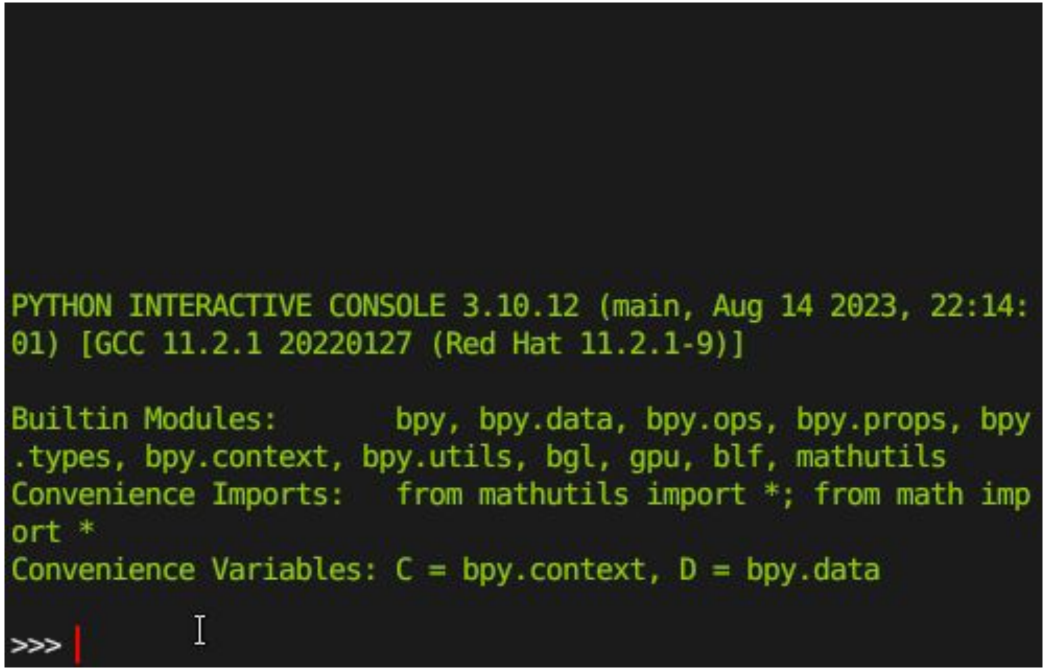
Resolution Scale 0.86

Line Width Default

Your First (But not last!) Blender 3D Python Script

Your First Script (1/2)

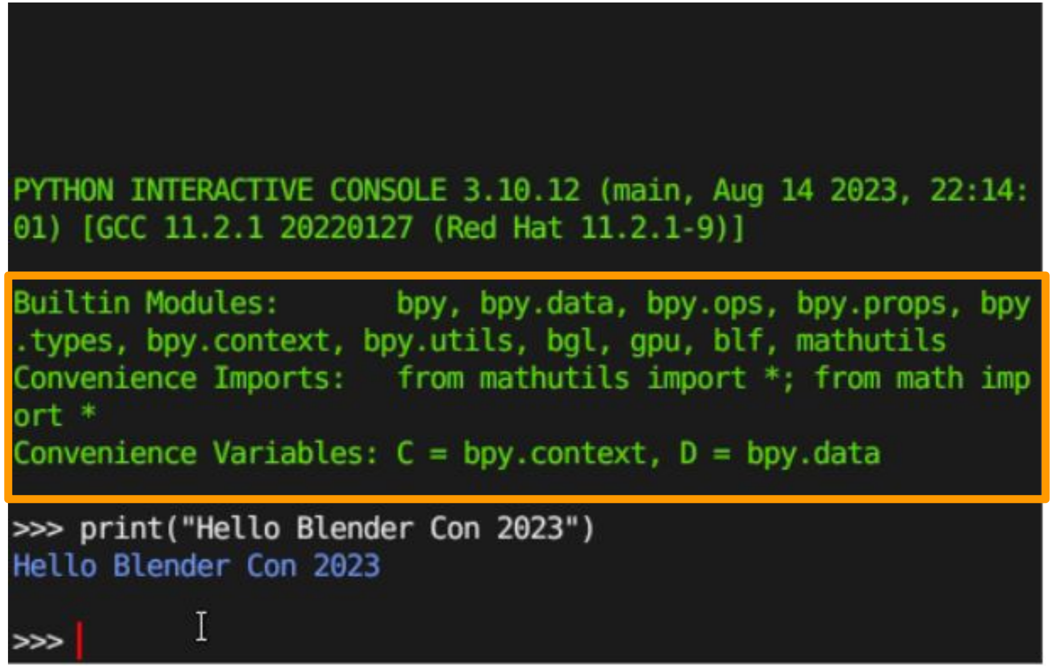
- From the **Python Console** you can type in your first command:
  - `print(“some_string”)`
    - This should output text to the console
- Wonderful -- congratulations on your first script!



```
PYTHON INTERACTIVE CONSOLE 3.10.12 (main, Aug 14 2023, 22:14:
01) [GCC 11.2.1 20220127 (Red Hat 11.2.1-9)]

Builtin Modules:       bpy, bpy.data, bpy.ops, bpy.props, bpy
.types, bpy.context, bpy.utils, bgl, gpu, blf, mathutils
Convenience Imports:   from mathutils import *; from math imp
ort *
Convenience Variables: C = bpy.context, D = bpy.data

>>> 
```

Your First Script (2/2)

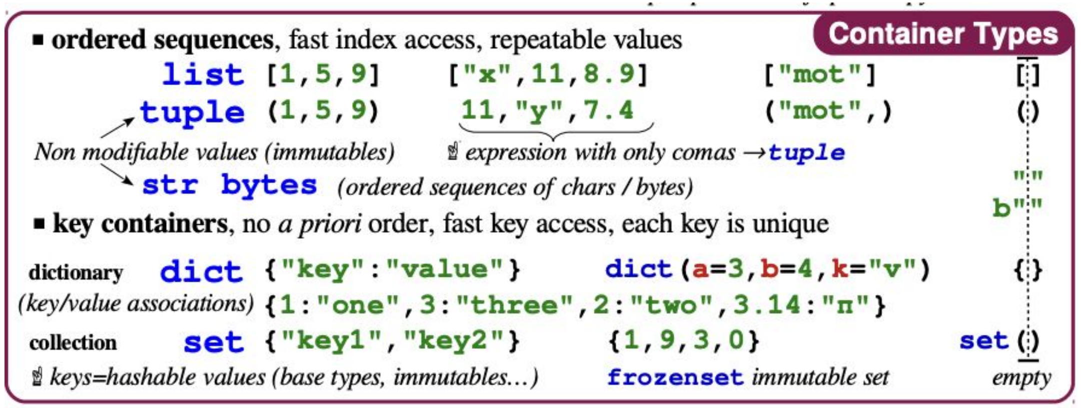
- We’ll talk a little bit about some of these important **Builtin Modules** throughout this talk.
  - Inside Blender, the python console, automatically loads these for us
  - Later on, in our scripts, we will manually import these modules.
- The other thing to note is that we are using Python `3.10.13`
  - Your version may differ, but you’ll want a relatively recent version of Python
    - i.e. Python 3.10.XX or greater is ideal moving forward



```
PYTHON INTERACTIVE CONSOLE 3.10.12 (main, Aug 14 2023, 22:14:
01) [GCC 11.2.1 20220127 (Red Hat 11.2.1-9)]

Builtin Modules:       bpy, bpy.data, bpy.ops, bpy.props, bpy
.types, bpy.context, bpy.utils, bgl, gpu, blf, mathutils
Convenience Imports:   from mathutils import *; from math imp
ort *
Convenience Variables: C = bpy.context, D = bpy.data

>>> print("Hello Blender Con 2023")
Hello Blender Con 2023

>>>
```

(Aside) Python Cheat Sheet

- I'll assume you have some amount of Python
  - Here's a brief cheat sheet on the right
- If you're pretty comfortable with:
  - lists, dictionaries, iteration, and classes you're all ready!



Container Types

▪ **ordered sequences**, fast index access, repeatable values

```
list  [1,5,9]    ["x",11,8.9]    ["mot"]    []
tuple (1,5,9)    11,"y",7.4      ("mot",)   ()
```

Non modifiable values (immutables) ☝ *expression with only comas* → **tuple**

```
str bytes
```
(ordered sequences of chars / bytes) "" b""

▪ **key containers**, no *a priori* order, fast key access, each key is unique

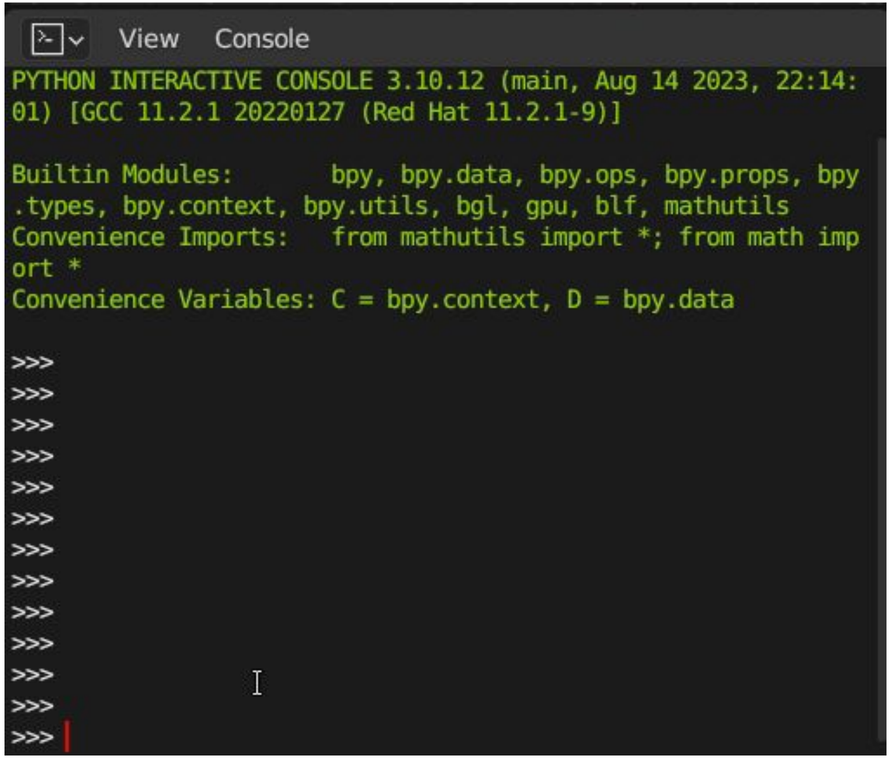
```
dictionary  dict {"key":"value"}      dict(a=3,b=4,k="v")   {}
(key/value associations) {1:"one",3:"three",2:"two",3.14:"π"}
collection  set {"key1","key2"}       {1,9,3,0}             set()
```

☝ *keys=hashable values (base types, immutables…)* **frozenset** *immutable set* *empty*

https://perso.limsi.fr/pointal/_media/python:cours:mementopython3-english.pdf

The Power of Python at Your Fingertips! (1/2)

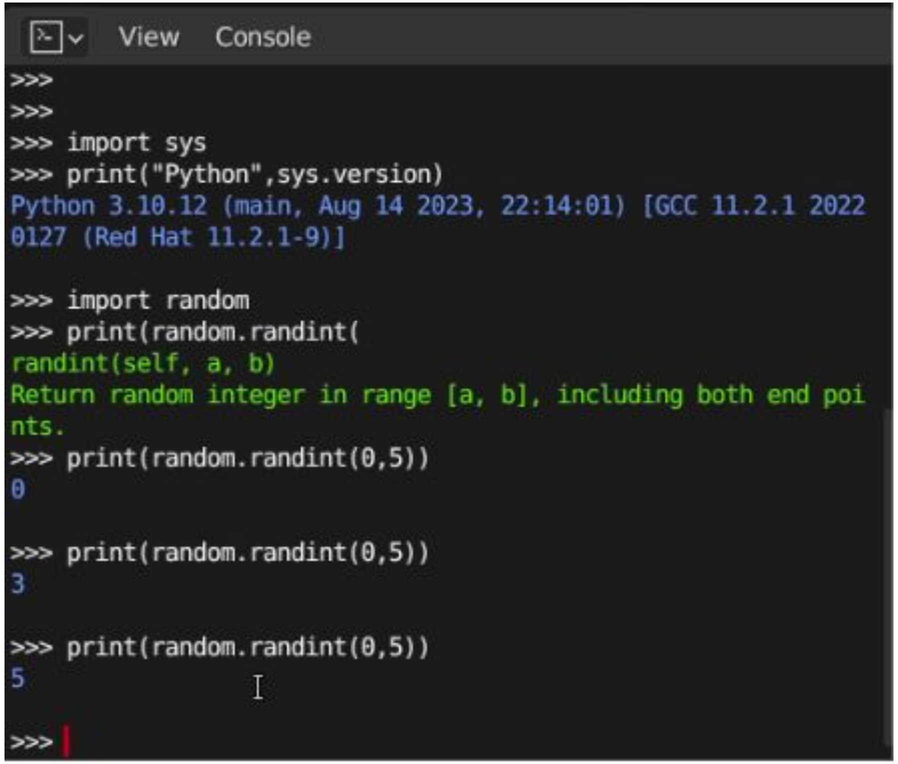
- At this point -- you have all of Python available!
    - The example to the right shows importing ‘sys’ and ‘random’ libraries.
    - The ‘`sys.version`’ tells us what version of Python we have (incase you missed it)
        - Generally speaking, we want Python version 3+ for this tutorial.
    - For printing the random numbers, try repeating the command a few times



```
PYTHON INTERACTIVE CONSOLE 3.10.12 (main, Aug 14 2023, 22:14:
01) [GCC 11.2.1 20220127 (Red Hat 11.2.1-9)]

Builtin Modules:       bpy, bpy.data, bpy.ops, bpy.props, bpy
.types, bpy.context, bpy.utils, bgl, gpu, blf, mathutils
Convenience Imports:   from mathutils import *; from math imp
ort *
Convenience Variables: C = bpy.context, D = bpy.data


>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
>>>
```

The Power of Python at Your Fingertips! (2/2)

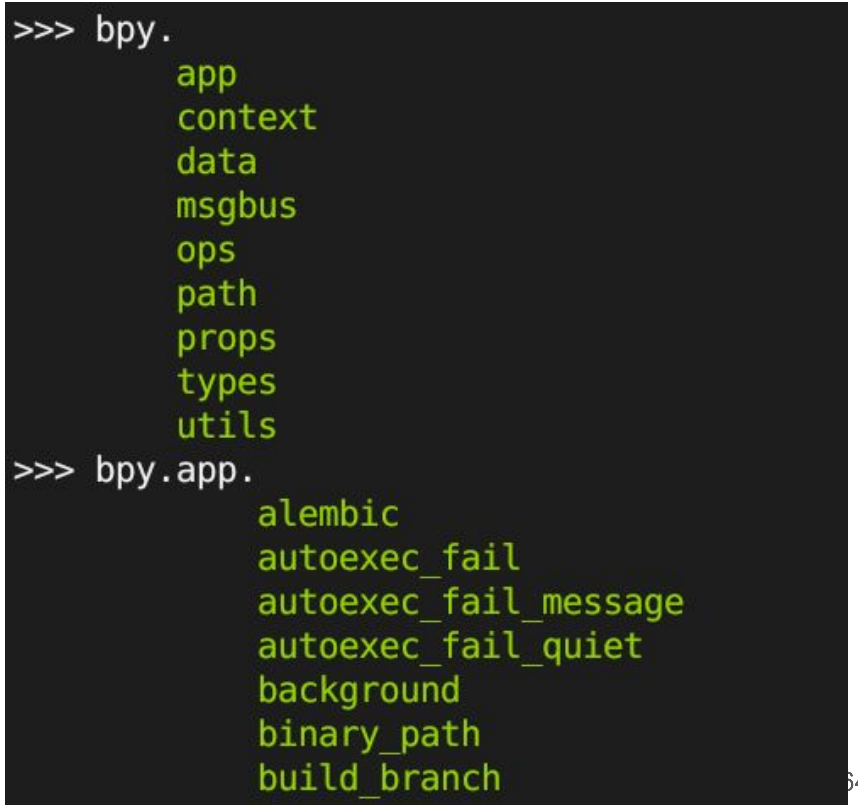
- (Still capture of the code)



```
>>>
>>>
>>> import sys
>>> print("Python",sys.version)
Python 3.10.12 (main, Aug 14 2023, 22:14:01) [GCC 11.2.1 2022
0127 (Red Hat 11.2.1-9)]

>>> import random
>>> print(random.randint(
randint(self, a, b)
Return random integer in range [a, b], including both end poi
nts.
>>> print(random.randint(0,5))
0

>>> print(random.randint(0,5))
3

>>> print(random.randint(0,5))
5

>>>
```

(Aside) Console Productivity Tip(s)

- To save yourself time, and re-execute a command, press the ‘up’ and ‘down’ arrow keys to cycle between your command history
    - Pressing ‘enter’ again will execute the command again (try with the random numbers)
- Use ‘tab’ to autocomplete text that you start typing.
    - This is a big time saver for typing out functions, variable names, etc.
        - As a learner, this is also useful for exploring which commands are available.



```
>>> bpy.
          app
          context
          data
          msgbus
          ops
          path
          props
          types
          utils
>>> bpy.app.
              alembic
              autoexec_fail
              autoexec_fail_message
              autoexec_fail_quiet
              background
              binary_path
              build_branch
```

Another Script -- Script for Timing

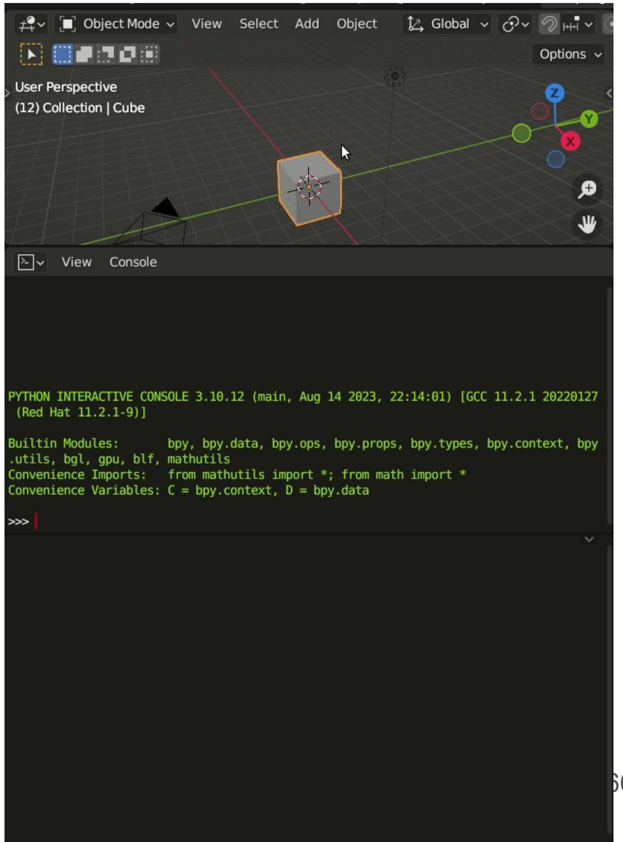
- Again, demonstrating usage of Python
  - It's nice feedback to the user
  - I've found if you have a timer for when the operation starts ,and some sort of reporting when the operation is finished
    - From a development standpoint -- it's useful for performance, and otherwise knowing when your operation is done as well
  - But -- it's not just Python that we have access to

```
>>>
>>>
>>>
>>>
>>>
>>>
```

```
# Text for copy & pasting
import time
start_timer = time.time()
print("elapsed",time.time() - start_timer)
```

The Power of Blender at Your Fingertips! (1/2)

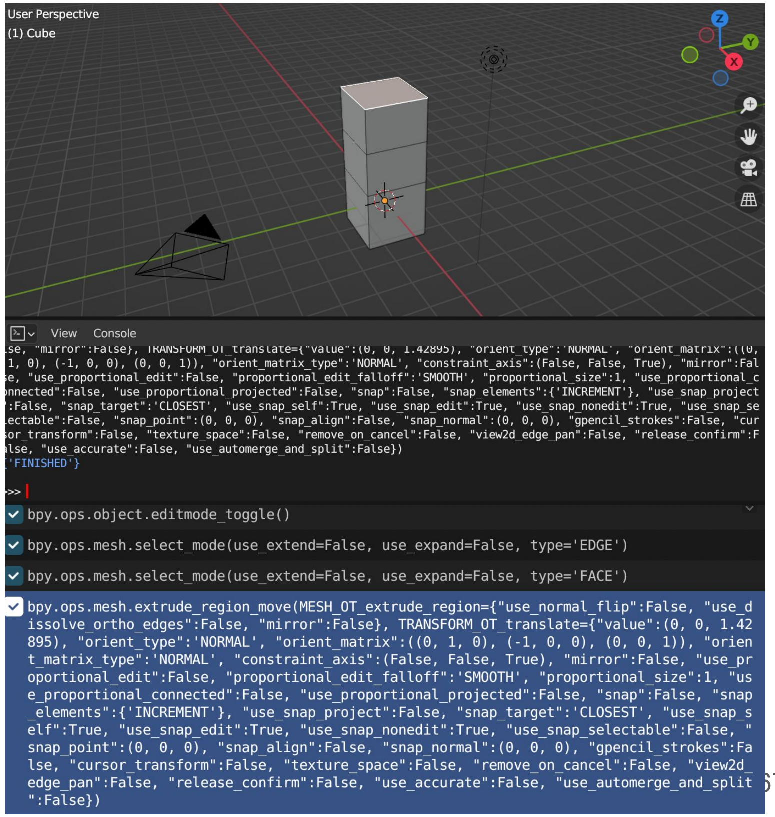
- What's also very neat about Blender 3D is learning some of the commands 'as you normally use blender'
    - Take a moment to modify the geometry of the cube
    - As you modify the geometry, you'll observe the **info log** is updated!
- Wow -- observe that we can see the script actions as they take place!



```
PYTHON INTERACTIVE CONSOLE 3.10.12 (main, Aug 14 2023, 22:14:01) [GCC 11.2.1 20220127
 (Red Hat 11.2.1-9)]

Builtin Modules:       bpy, bpy.data, bpy.ops, bpy.props, bpy.types, bpy.context, bpy
.utils, bgl, gpu, blf, mathutils
Convenience Imports:   from mathutils import *; from math import *
Convenience Variables: C = bpy.context, D = bpy.data

>>>
```

The Power of Blender at Your Fingertips! (2/2)

- Exercise: Try copying and pasting the previous ‘extrude’ command from the **info log**, into the **python console** which repeats the extrude of the selected face.

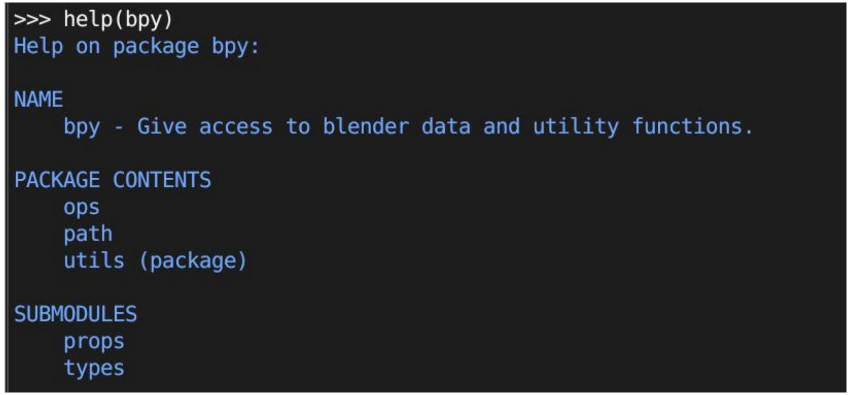


```
se, "mirror":False}, TRANSFORM_OT_translate={"value":(0, 0, 1.42895), "orient_type":'NORMAL', "orient_matrix":((0,
1, 0), (-1, 0, 0), (0, 0, 1)), "orient_matrix_type":'NORMAL', "constraint_axis":(False, False, True), "mirror":Fal
se, "use_proportional_edit":False, "proportional_edit_falloff":'SMOOTH', "proportional_size":1, "use_proportional_c
nnected":False, "use_proportional_projected":False, "snap":False, "snap_elements":{'INCREMENT'}, "use_snap_project
":False, "snap_target":'CLOSEST', "use_snap_self":True, "use_snap_edit":True, "use_snap_nonedit":True, "use_snap_se
lectable":False, "snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "gpencil_strokes":False, "cur
sor_transform":False, "texture_space":False, "remove_on_cancel":False, "view2d_edge_pan":False, "release_confirm":F
alse, "use_accurate":False, "use_automerge_and_split":False})
{'FINISHED'}

>>>
```

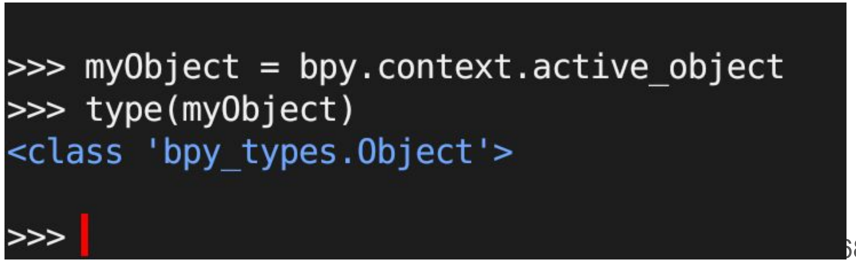
```
bpy.ops.object.editmode_toggle()

bpy.ops.mesh.select_mode(use_extend=False, use_expand=False, type='EDGE')

bpy.ops.mesh.select_mode(use_extend=False, use_expand=False, type='FACE')

bpy.ops.mesh.extrude_region_move(MESH_OT_extrude_region={"use_normal_flip":False, "use_d
issolve_ortho_edges":False, "mirror":False}, TRANSFORM_OT_translate={"value":(0, 0, 1.42
895), "orient_type":'NORMAL', "orient_matrix":((0, 1, 0), (-1, 0, 0), (0, 0, 1)), "orien
t_matrix_type":'NORMAL', "constraint_axis":(False, False, True), "mirror":False, "use_pr
oportional_edit":False, "proportional_edit_falloff":'SMOOTH', "proportional_size":1, "us
e_proportional_connected":False, "use_proportional_projected":False, "snap":False, "snap
_elements":{'INCREMENT'}, "use_snap_project":False, "snap_target":'CLOSEST', "use_snap_s
elf":True, "use_snap_edit":True, "use_snap_nonedit":True, "use_snap_selectable":False, "
snap_point":(0, 0, 0), "snap_align":False, "snap_normal":(0, 0, 0), "gpencil_strokes":Fa
lse, "cursor_transform":False, "texture_space":False, "remove_on_cancel":False, "view2d_
edge_pan":False, "release_confirm":False, "use_accurate":False, "use_automerge_and_split
":False})
```

Getting Help on Your Journey (1/2)

- **help(...)**
  - From the **Python Console** you can type 'help' on any module, function, class, or even a variable.
  - Exercise: Try help(bpy), help(bpy.data)
    - (Remember, these modules have been imported for us already)
    - You can use 'help' on any module to start exploring some of the 'classes' and 'functions' available.
- **type(...)**
  - This is useful for querying the type of something that has already been created.



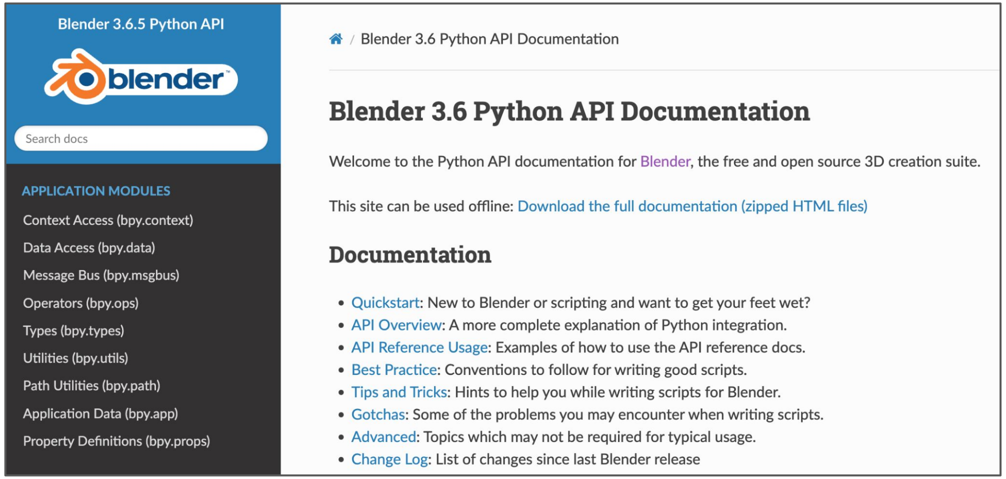
```
>>> help(bpy)
Help on package bpy:

NAME
    bpy - Give access to blender data and utility functions.

PACKAGE CONTENTS
    ops
    path
    utils (package)

SUBMODULES
    props
    types
```



```
>>> myObject = bpy.context.active_object
>>> type(myObject)
<class 'bpy_types.Object'>

>>> 
```

Getting Help on Your Journey (2/2)

- The Python API Documentation online is a great resource
  - Note: I recommend downloading a copy to be used offline for faster browsing
    - (Also useful for long airplane rides :))


Blender 3.6.5 Python API

Search docs

APPLICATION MODULES

Context Access (bpy.context)
Data Access (bpy.data)
Message Bus (bpy.msgbus)
Operators (bpy.ops)
Types (bpy.types)
Utilities (bpy.utils)
Path Utilities (bpy.path)
Application Data (bpy.app)
Property Definitions (bpy.props)

/ Blender 3.6 Python API Documentation

Blender 3.6 Python API Documentation

Welcome to the Python API documentation for Blender, the free and open source 3D creation suite.

This site can be used offline: Download the full documentation (zipped HTML files)

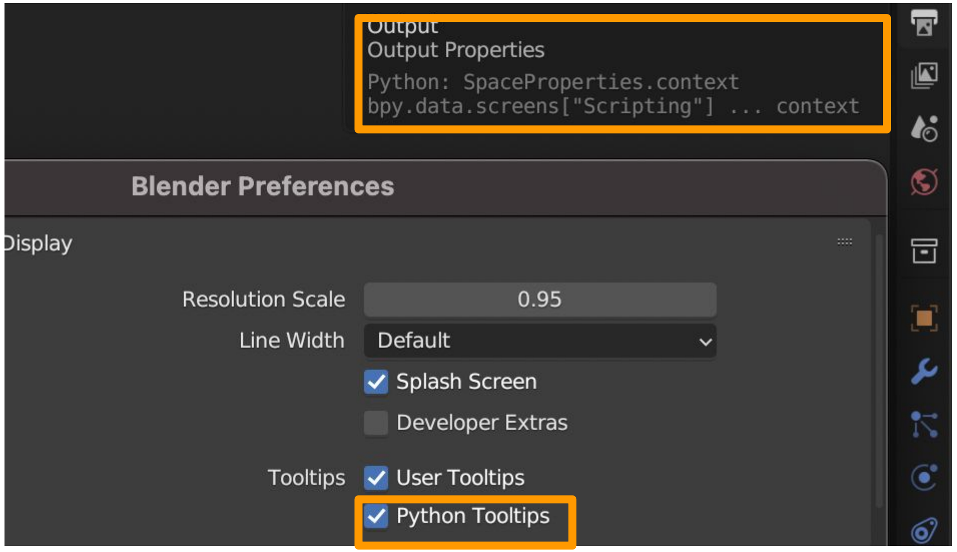
Documentation

- Quickstart: New to Blender or scripting and want to get your feet wet?
- API Overview: A more complete explanation of Python integration.
- API Reference Usage: Examples of how to use the API reference docs.
- Best Practice: Conventions to follow for writing good scripts.
- Tips and Tricks: Hints to help you while writing scripts for Blender.
- Gotchas: Some of the problems you may encounter when writing scripts.
- Advanced: Topics which may not be required for typical usage.
- Change Log: List of changes since last Blender release

https://docs.blender.org/api/current/index.html

Enabling ‘Python Tooltips’ for Developers

- Another very useful way to explore the Python API is to **enable ‘Python Tooltips’**
  - This is done in the ‘preferences’ modal.
  - Enabling ‘Python Tooltips’ will show you additional information about various tools you are use to clicking on -- and guide you to the python API.
    - (See example in the top-right)


Output
Output Properties
```
Python: SpaceProperties.context
bpy.data.screens["Scripting"] ... context
```
Blender Preferences

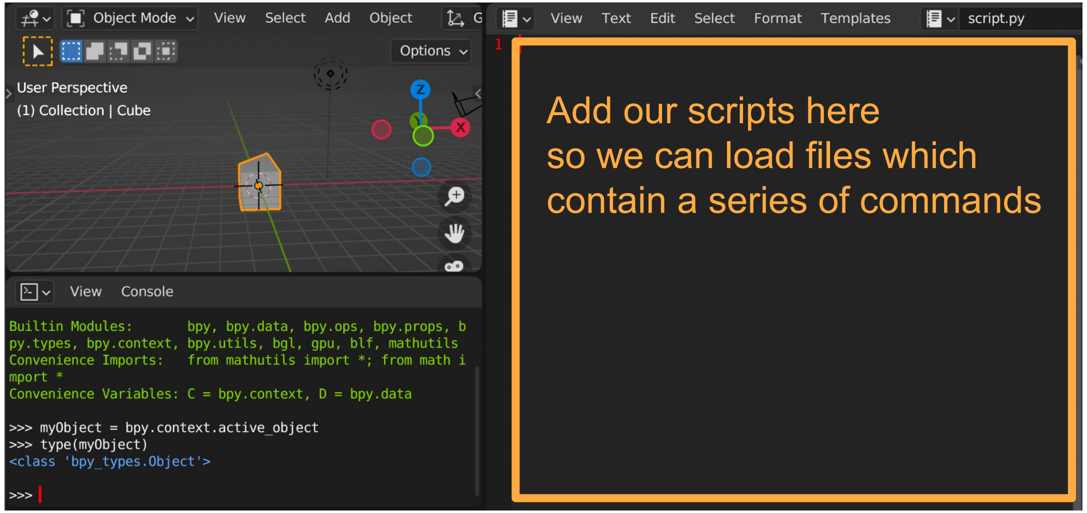
Display

Resolution Scale: 0.95
Line Width: Default
[✓] Splash Screen
[ ] Developer Extras
Tooltips: [✓] User Tooltips
[✓] Python Tooltips

Using the Internal Text Editor

Blender 3D Internal Text Editor

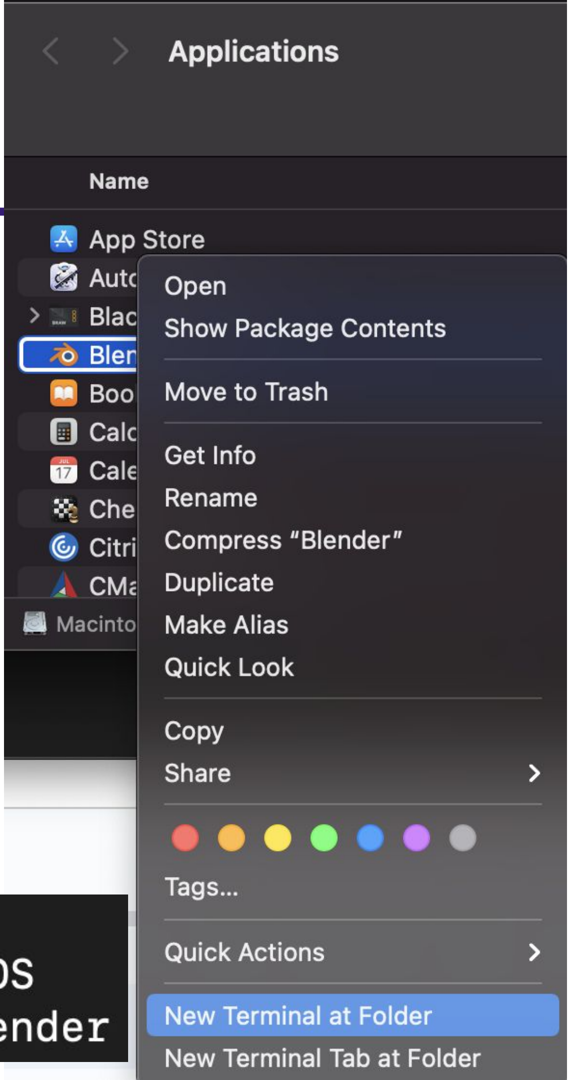
- At some point we likely will want to be able to create larger scripts that execute a series of commands to perform some work.
  - For this presentation we'll write our scripts in the Text Editor
  - Note: You can use your favorite Text Editor (VIM, VSCode, etc.) to also write your scripts.

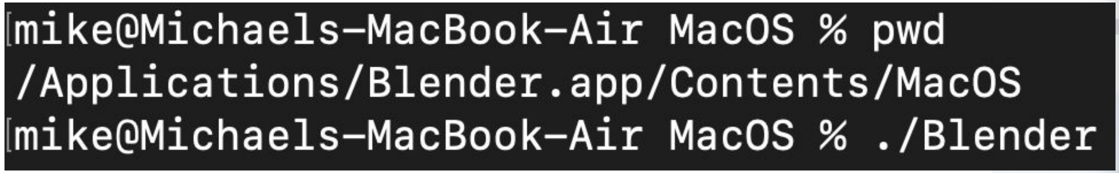


```
Builtin Modules:       bpy, bpy.data, bpy.ops, bpy.props, b
py.types, bpy.context, bpy.utils, bgl, gpu, blf, mathutils
Convenience Imports:   from mathutils import *; from math i
mport *
Convenience Variables: C = bpy.context, D = bpy.data

>>> myObject = bpy.context.active_object
>>> type(myObject)
<class 'bpy_types.Object'>

>>> 
```

Tip: Launch Blender from Terminal

- In order to help us debug and 'print' out text, it is most useful to launch Blender from the terminal.
  - Then when we execute our scripts we will get text output on the terminal where we launched.
  - On Mac
    - You will then use 'Option + P' to run your script
  - On Linux
    - You will use 'alt+p'





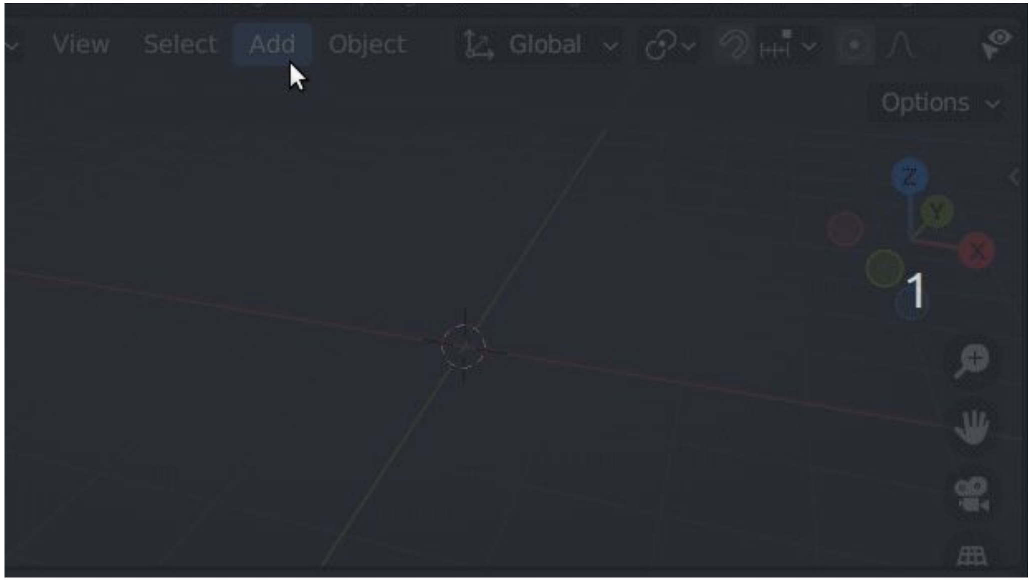
```
mike@Michaels-MacBook-Air MacOS % pwd
/Applications/Blender.app/Contents/MacOS
mike@Michaels-MacBook-Air MacOS % ./Blender
```

Solving the Bounding Box Problem with Python Scripting

Gathering our Tools from the Python API

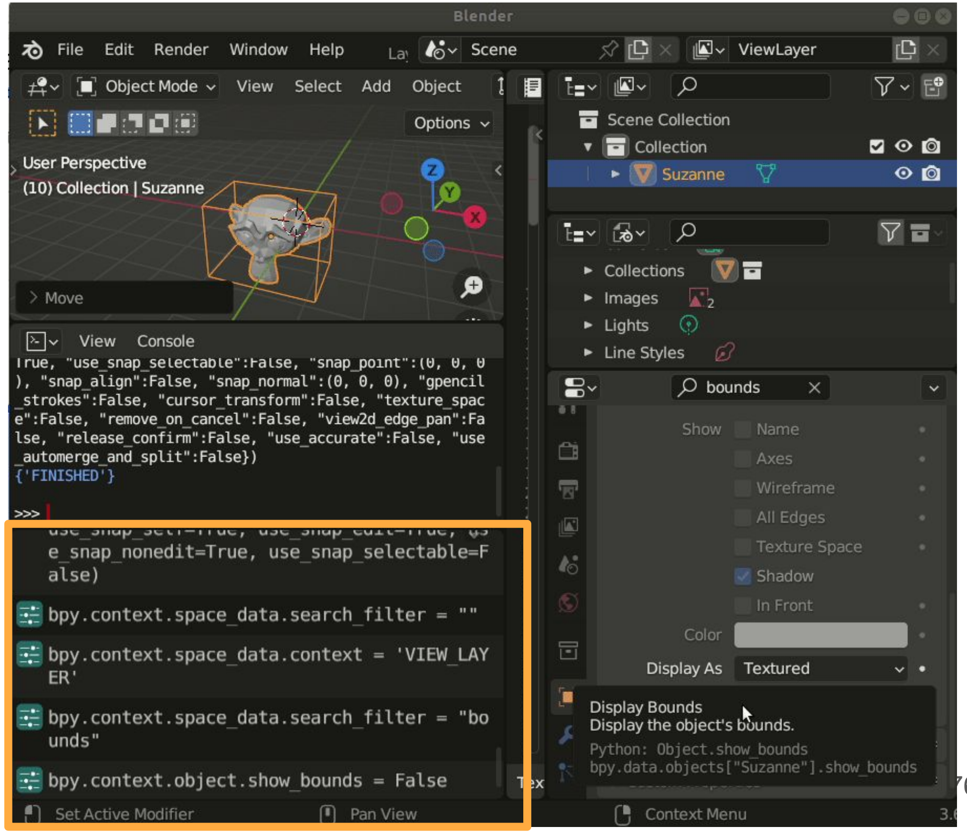
Creating a Bounding Box Programmatically

- So as was shown at the start of the talk, let’s begin our journey creating a bounding box
  - Now this is something that Blender3D already has the capability to do
  - However, learning how to do so from scratch will expose us to Blender’s API through Python.



Creating a Bounding Box Programmatically -- built-in

- Now of course *you could* call:
  - `.show.bounds = True`
  - That's not really in the spirit of this assignment...
- However, this does introduce the 'bpy.context' (see the bottom-left of info log) module which is of use



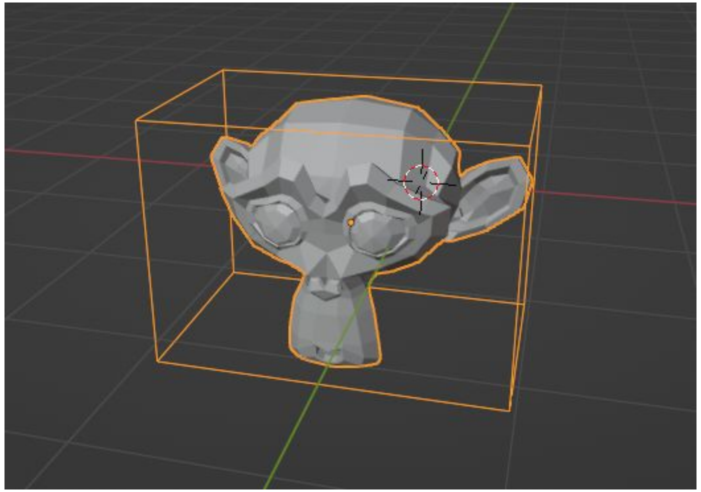
```
True, "use_snap_selectable":False, "snap_point":(0, 0, 0
), "snap_align":False, "snap_normal":(0, 0, 0), "gpencil
_strokes":False, "cursor_transform":False, "texture_spac
e":False, "remove_on_cancel":False, "view2d_edge_pan":Fa
lse, "release_confirm":False, "use_accurate":False, "use
_automerge_and_split":False})
{'FINISHED'}

>>>
```

```
e_snap_nonedit=True, use_snap_selectable=F
alse)
bpy.context.space_data.search_filter = ""
bpy.context.space_data.context = 'VIEW_LAY
ER'
bpy.context.space_data.search_filter = "bo
unds"
bpy.context.object.show_bounds = False
```

Display Bounds
Display the object's bounds.
Python: Object.show_bounds
bpy.data.objects["Suzanne"].show_bounds

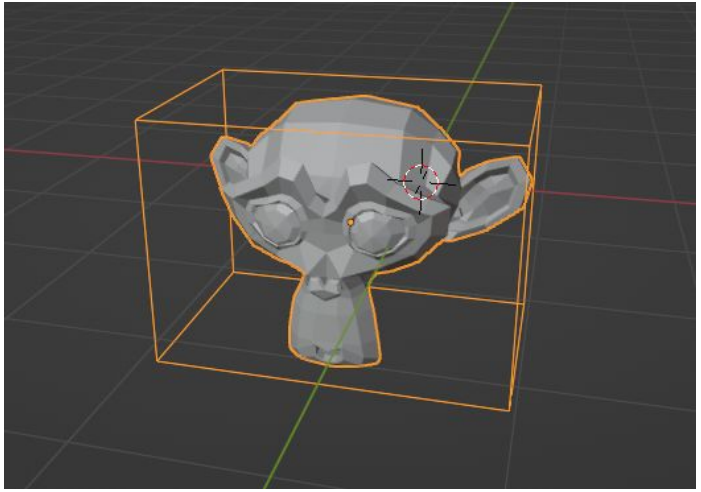
Exercise: How do you compute the bounding box? (1/2)

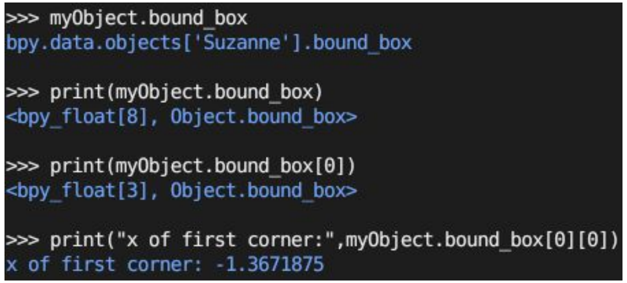
- Now if you had to compute the bounding box from scratch -- how would you do it?
  - (If you're watching this in the future you can pause the video and write out a solution)
  - For my current audience, I'm going to forward us to one solution -- there's a couple ways to approach this



Exercise: How do you compute the bounding box? (2/2)

- Simplest solution
  - Iterate through all of the vertices
    - Keep track of both the minimum and maximum x,y,z values
- Another solution for obtaining the bounds is to otherwise use:
  - `myObject.bound_box`
  - This returns the '8' vertices of the bounding box
- (Aside: This is an axis-aligned bounding box, but we can apply a transform to get an oriented-bounding box)

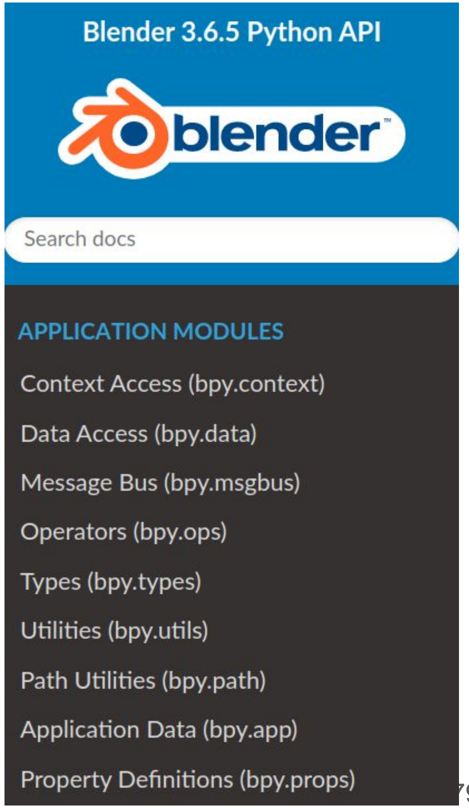




```
>>> myObject.bound_box
bpy.data.objects['Suzanne'].bound_box

>>> print(myObject.bound_box)
<bpy_float[8], Object.bound_box>

>>> print(myObject.bound_box[0])
<bpy_float[3], Object.bound_box>

>>> print("x of first corner:",myObject.bound_box[0][0])
x of first corner: -1.3671875
```

A few Blender Python (bpy) modules of Importance (1/2)

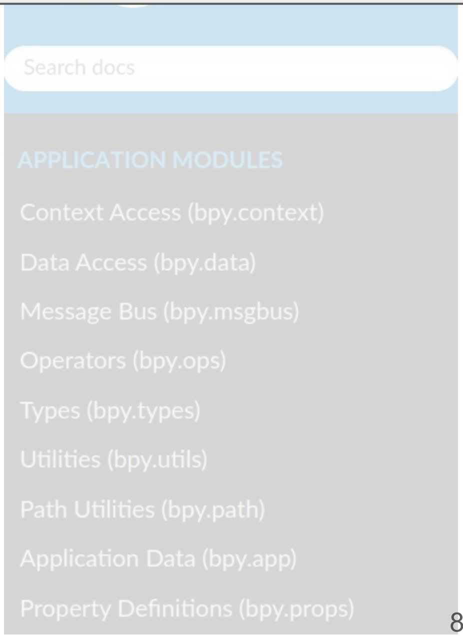
- `import bpy`
  - This is the main module of the programming interface in Blender.
- `import bpy.context`
  - This module captures the current state of the user interaction
    - (e.g. selection or current mode)
  - Note: This is often aliased as ‘C’ for ‘bpy.context’
- `import bpy.data`
  - This is the storage of blender objects
    - Anything found within bpy.data.objects is something that can be displayed in the Blender 3D viewport
      - (e.g. camera, lights, curves, meshes, etc.)
  - Note: This is often aliased to ‘D’ for ‘bpy.data’
- `import bpy.ops`
  - Functions that can be invoked in the interface



A few Blender Python (bpy

- `import bpy`
  - This is the main m the progr
- **`import bpy.context`**
  - This module captures the current state of the user interaction
    - (e.g. selection or current mode)
  - Note: This is often aliased as ‘C’ for ‘bpy.context’
- **`import bpy.data`**
  - This is the storage of blender objects
    - Anything found within bpy.data.objects is something that can be displayed in the Blender 3D viewport
      - (e.g. camera, lights, curves, meshes, etc.)
  - Note: This is often aliased to ‘D’ for ‘bpy.data’
- `import bpy.ops`
  - Functions that can be invoked in the interface

- Both of these modules are going to be important for us to work in
  - One for selecting our object of interest
  - The second for getting data



Search docs

APPLICATION MODULES

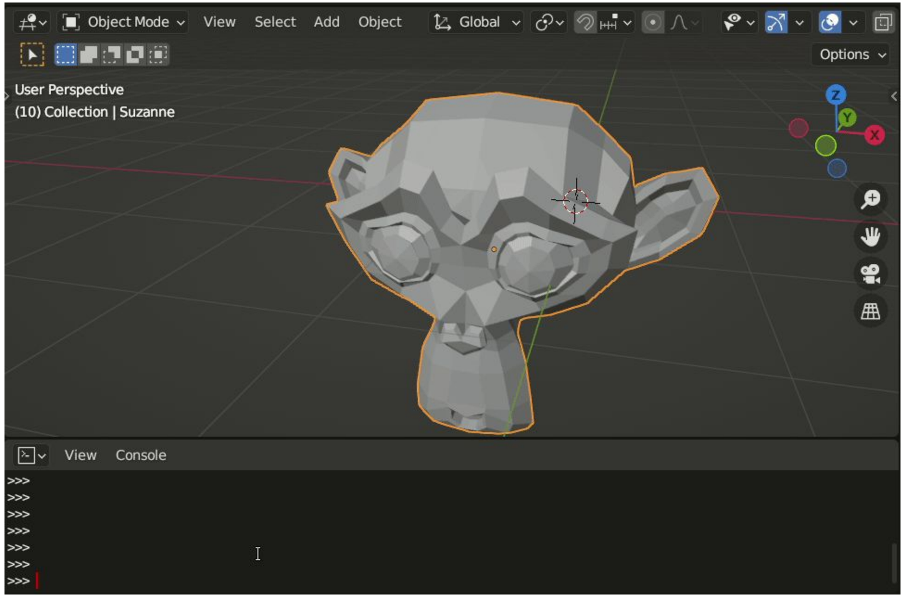
Context Access (bpy.context)
Data Access (bpy.data)
Message Bus (bpy.msgbus)
Operators (bpy.ops)
Types (bpy.types)
Utilities (bpy.utils)
Path Utilities (bpy.path)
Application Data (bpy.app)
Property Definitions (bpy.props)

Bounding Box

Implementation

`bpy.context` and selecting the current object

- So again the **`bpy.context`** is useful for telling us what is going on in an ‘area’ of our screen.
- Usually these are ‘read-only’ types of things we can get
  - But it’s very useful for instance if we want to store a variable to our currently selected object
- e.g.
  - `myObject = bpy.context.active_object`

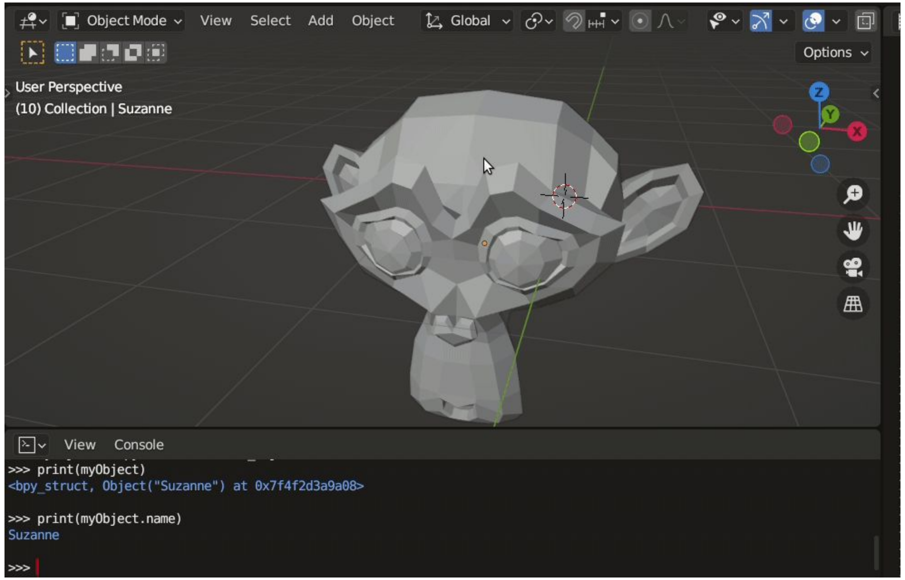


```
>>>
>>>
>>>
>>>
>>>
>>>
>>>
```

Demonstrates getting a handle to the active object

Acquiring the Geometry of our current object

- As we know, 3D objects are often defined by:
  - vertices, edges, and polygons (3 or more edges)
- # Now let's acquire some data
  - `verts = myObject.data.vertices`
  - `edges = myObject.data.edges`
  - `faces = myObject.data.polygons`
- Note: When we access an objects ‘data’, sometimes you’ll hear this referred to as a **data-block**

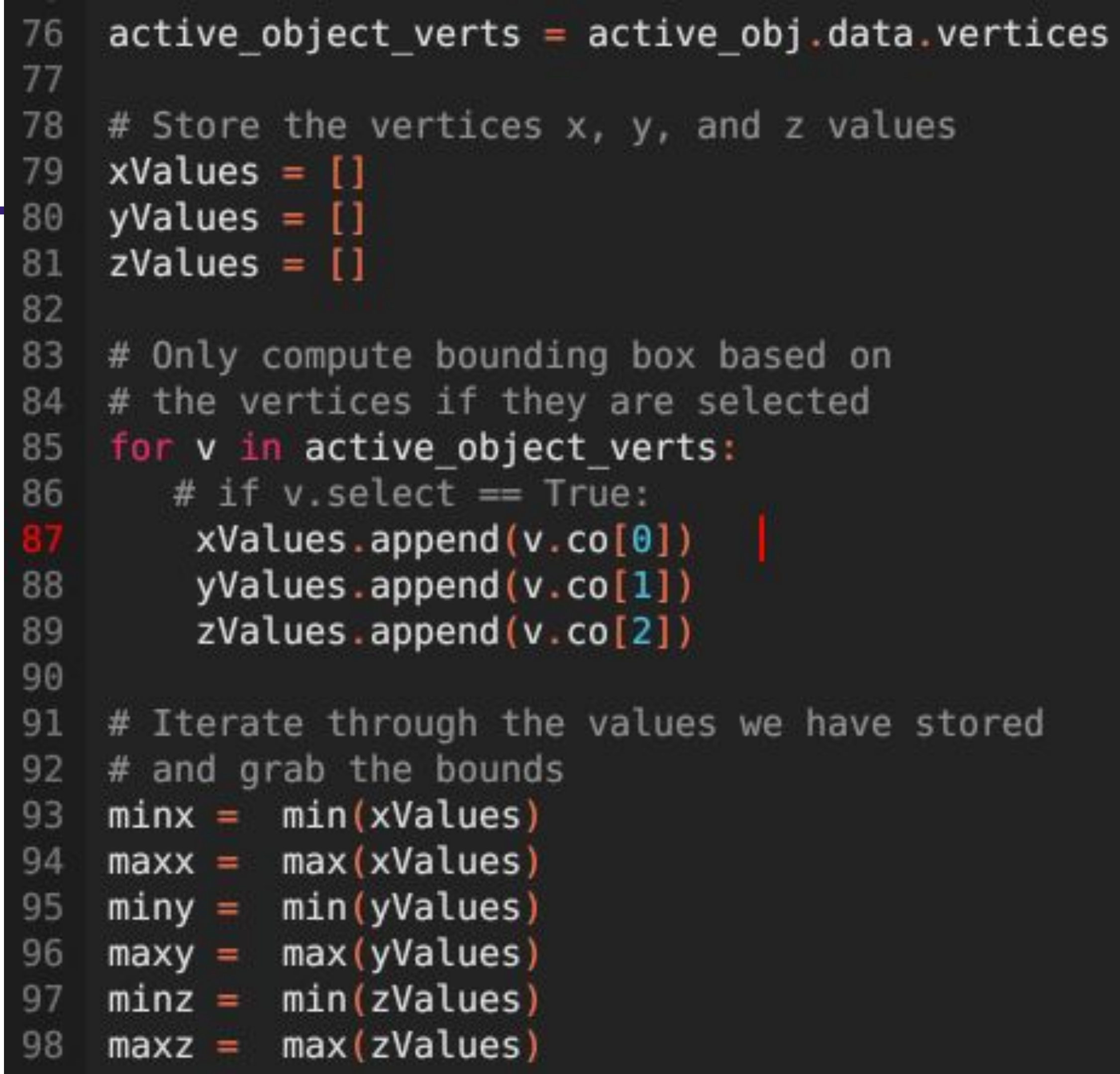


```
>>> print(myObject)
<bpy_struct, Object("Suzanne") at 0x7f4f2d3a9a08>

>>> print(myObject.name)
Suzanne

>>>
```

Demonstrates getting vertex, edge, and polygon information

Computing the Bounds (1/3)

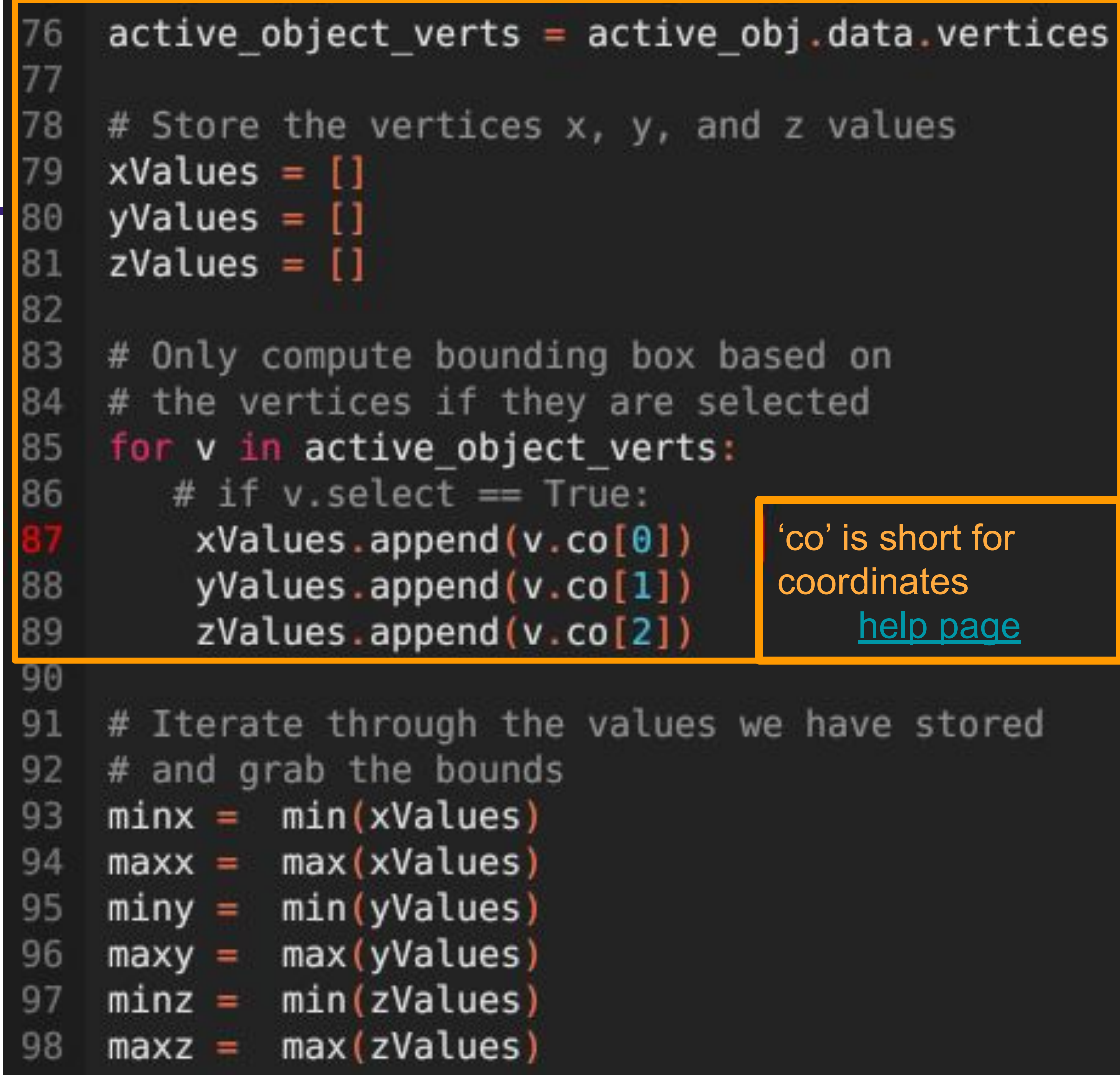
- I’ve opted for as simple of an algorithm as possible



```
active_object_verts = active_obj.data.vertices

# Store the vertices x, y, and z values
xValues = []
yValues = []
zValues = []

# Only compute bounding box based on
# the vertices if they are selected
for v in active_object_verts:
   # if v.select == True:
    xValues.append(v.co[0])
    yValues.append(v.co[1])
    zValues.append(v.co[2])

# Iterate through the values we have stored
# and grab the bounds
minx =  min(xValues)
maxx =  max(xValues)
miny =  min(yValues)
maxy =  max(yValues)
minz =  min(zValues)
maxz =  max(zValues)
```

Computing the Bounds (2/3)

- First grab the vertices
  - We're going to want our own 'List' of vertices to work with (and later generate some geometry)
  - Note: I have commented out to only compute bounding box on selected vertices (line 87) -- try to play around with that on your own time ;)
    - Hint: May or may not need to be in edit mode.

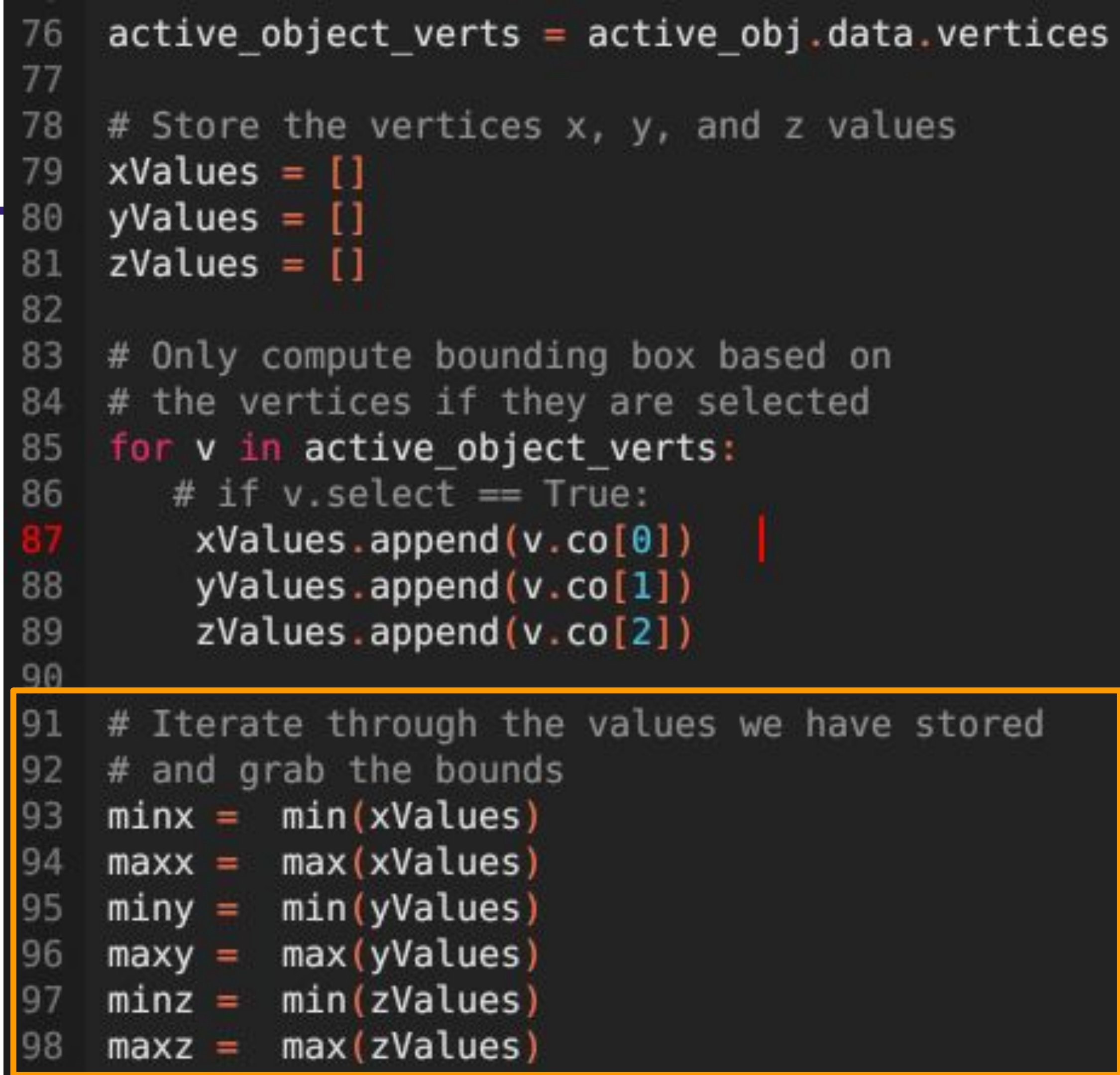


```
76  active_object_verts = active_obj.data.vertices
77
78  # Store the vertices x, y, and z values
79  xValues = []
80  yValues = []
81  zValues = []
82
83  # Only compute bounding box based on
84  # the vertices if they are selected
85  for v in active_object_verts:
86      # if v.select == True:
87          xValues.append(v.co[0])
88          yValues.append(v.co[1])
89          zValues.append(v.co[2])
90
91  # Iterate through the values we have stored
92  # and grab the bounds
93  minx =  min(xValues)
94  maxx =  max(xValues)
95  miny =  min(yValues)
96  maxy =  max(yValues)
97  minz =  min(zValues)
98  maxz =  max(zValues)
```

'co' is short for coordinates
help page

Computing the Bounds (3/3)

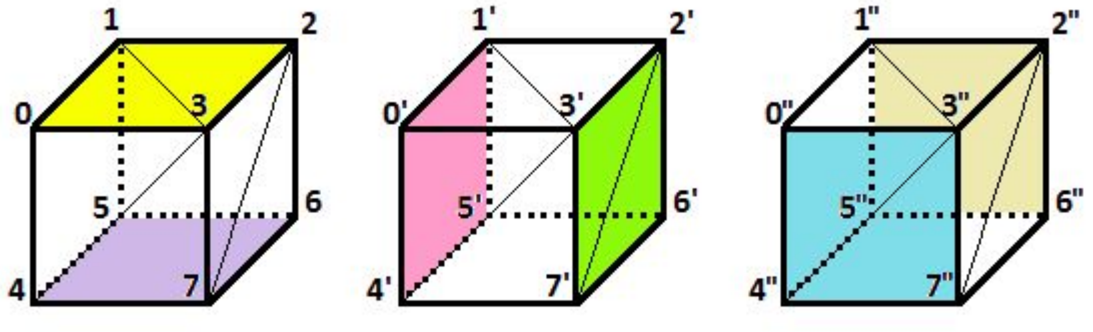
- Finally, compute the bounds
  - The min and max functions are useful here for searching through a range



```python
active_object_verts = active_obj.data.vertices

# Store the vertices x, y, and z values
xValues = []
yValues = []
zValues = []

# Only compute bounding box based on
# the vertices if they are selected
for v in active_object_verts:
    # if v.select == True:
      xValues.append(v.co[0])
      yValues.append(v.co[1])
      zValues.append(v.co[2])

# Iterate through the values we have stored
# and grab the bounds
minx =  min(xValues)
maxx =  max(xValues)
miny =  min(yValues)
maxy =  max(yValues)
minz =  min(zValues)
maxz =  max(zValues)
```

Creating a ‘Bounding Box’ (1/2)

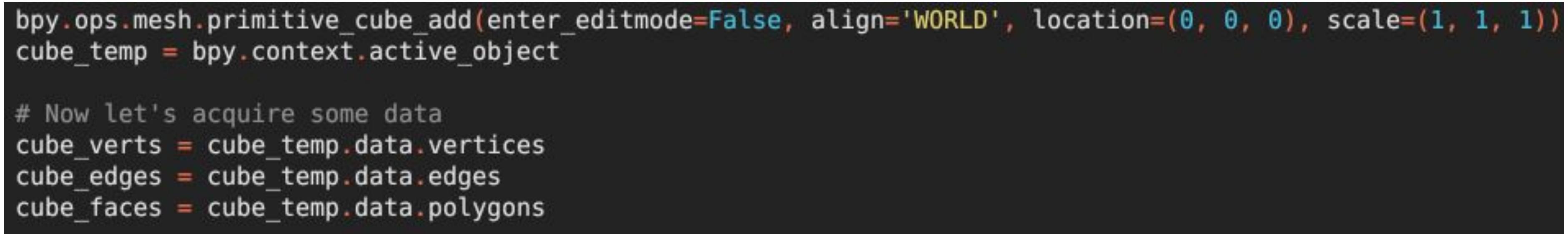
- Now that we have the boundaries, we need to create a ‘box’ object
- In order to generate a ‘mesh’ we have a few choices
  - Some folks with graphics programming, may go ahead and want to create the ‘indexed cube’ and calculate the vertices, edges, and polygons (with the correct winding order)



https://alg.manifoldapp.org/api/proxy/ingestion_sources/85b8a903-6d37-4a26-b376-552ce0ef528b

Creating a ‘Bounding Box’ (2/2)

- Now that we have the boundaries, we need to create a ‘box’ object
- In order to generate a ‘mesh’ we have a few choices
  - Some folks with graphics programming, may go ahead and want to create the ‘indexed cube’ and calculate the vertices, edges, and polygons (with the correct winding order)
  - A second choice, is to simply generate a cube from blender, and reposition the corner vertices
    - This does the hard work of preserving the connectivity for us.



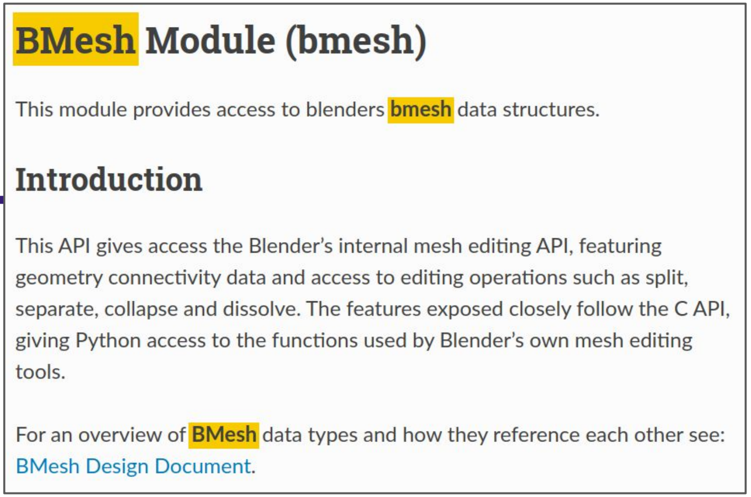
```
bpy.ops.mesh.primitive_cube_add(enter_editmode=False, align='WORLD', location=(0, 0, 0), scale=(1, 1, 1))
cube_temp = bpy.context.active_object

# Now let's acquire some data
cube_verts = cube_temp.data.vertices
cube_edges = cube_temp.data.edges
cube_faces = cube_temp.data.polygons
```

bmesh

Blender Mesh Format

BMesh (bmesh)

- The BMesh API allows us to work with the internal mesh editing tools in blender.
  - i.e. Basically any operations that you'd like
- Probably most important for us is to just be able to grab data and put it into a mesh.
- There might come a time where you want to perform more interesting operations
  - (Note: When you run a script, you lock the mesh by operating on it, modify the mesh, and returns control to a user)



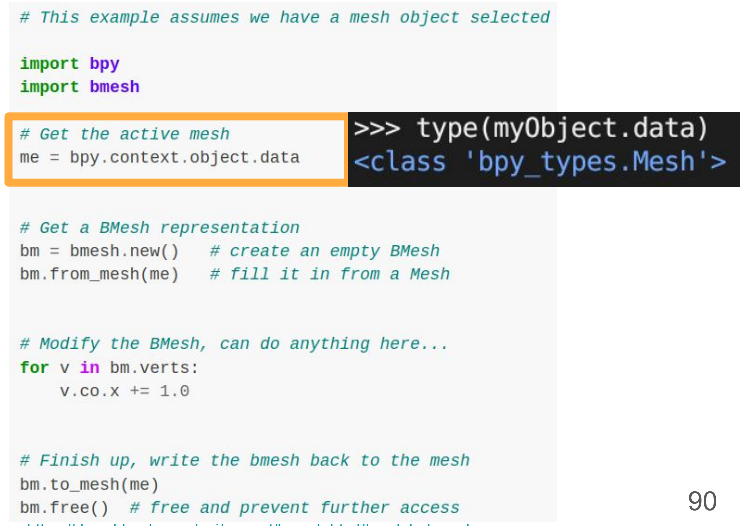
BMesh Module (bmesh)

This module provides access to blenders bmesh data structures.

Introduction

This API gives access the Blender's internal mesh editing API, featuring geometry connectivity data and access to editing operations such as split, separate, collapse and dissolve. The features exposed closely follow the C API, giving Python access to the functions used by Blender's own mesh editing tools.

For an overview of BMesh data types and how they reference each other see: BMesh Design Document.



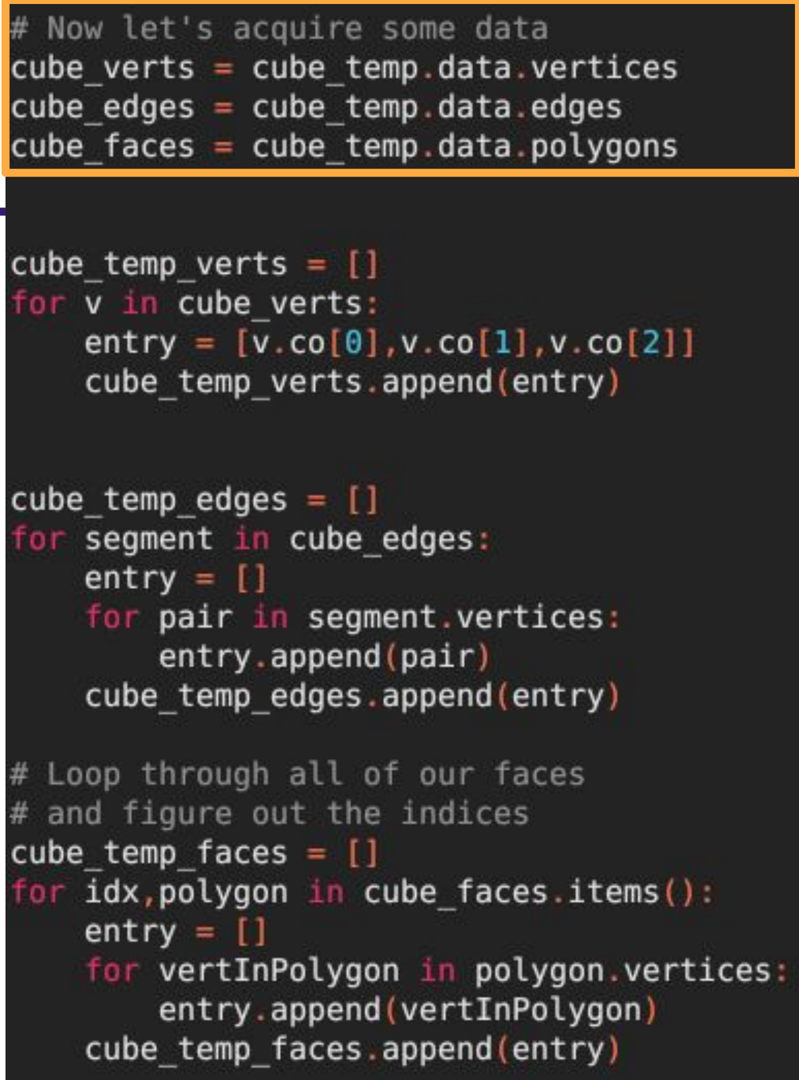
```
# This example assumes we have a mesh object selected

import bpy
import bmesh

# Get the active mesh
me = bpy.context.object.data


# Get a BMesh representation
bm = bmesh.new()   # create an empty BMesh
bm.from_mesh(me)   # fill it in from a Mesh


# Modify the BMesh, can do anything here...
for v in bm.verts:
    v.co.x += 1.0


# Finish up, write the bmesh back to the mesh
bm.to_mesh(me)
bm.free()  # free and prevent further access
```

```
>>> type(myObject.data)
<class 'bpy_types.Mesh'>
```

https://docs.blender.org/api/current/bmesh.html#module-bmesh

Iterating through data

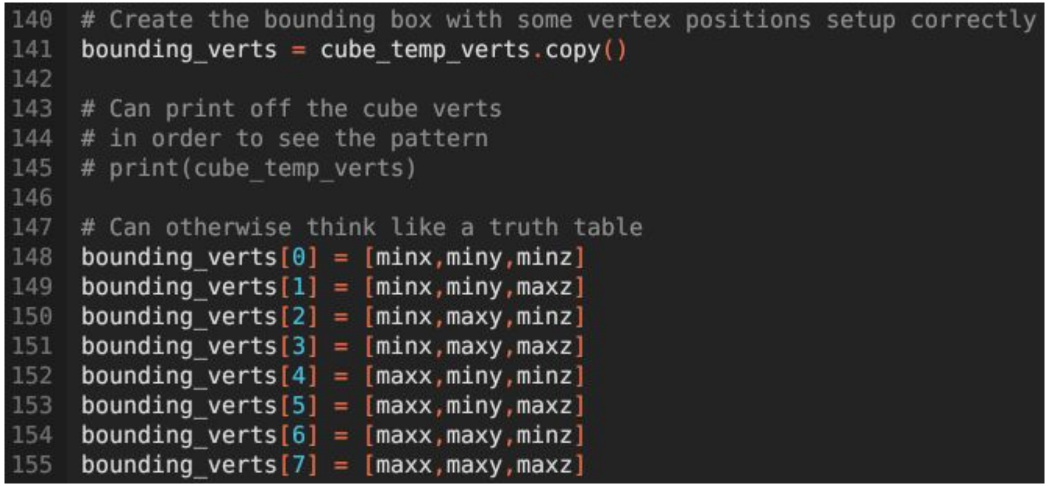
- It’s useful for us to store the vertices, edges, and faces in our own data structure to generate ‘a new mesh’ for our bounding box
    - The code below demonstrates how to ‘iterate’ through each of vertices, edges, and ‘polygons’ (i.e. faces)
        - Please be careful as to **not modify** the original data -- observe we are copying into our own list
        - Modifying a data structure while iterating could be unsafe
            - (‘search iterator invalidation’)
- Note: These blocks of code could be condensed further -- optimize at your level of Python!
    - List comprehension, unzip list, etc.



```
# Now let's acquire some data
cube_verts = cube_temp.data.vertices
cube_edges = cube_temp.data.edges
cube_faces = cube_temp.data.polygons


cube_temp_verts = []
for v in cube_verts:
    entry = [v.co[0],v.co[1],v.co[2]]
    cube_temp_verts.append(entry)


cube_temp_edges = []
for segment in cube_edges:
    entry = []
    for pair in segment.vertices:
        entry.append(pair)
    cube_temp_edges.append(entry)

# Loop through all of our faces
# and figure out the indices
cube_temp_faces = []
for idx,polygon in cube_faces.items():
    entry = []
    for vertInPolygon in polygon.vertices:
        entry.append(vertInPolygon)
    cube_temp_faces.append(entry)
```

Building Our Bounding Box

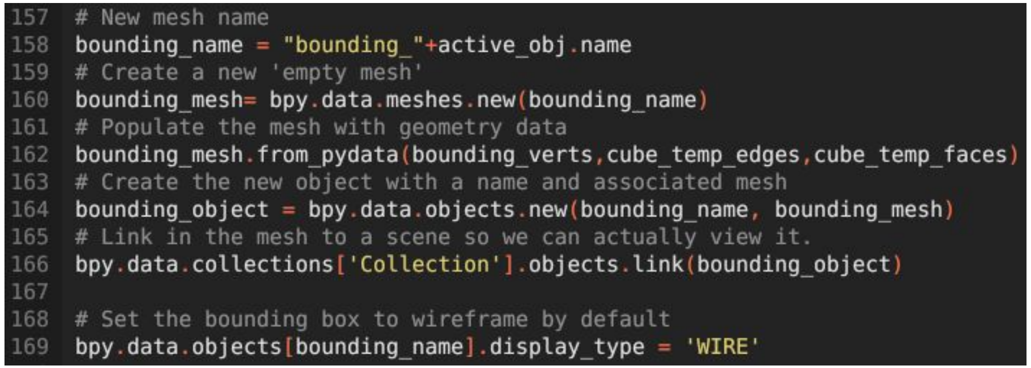
- Here is the little hack where I just need to reassign the vertices of our 'cube'
    - There's a pattern here you can follow
        - (Hint: It happens look like a truth table if you have taken a logic or discrete math subject)



```python
# Create the bounding box with some vertex positions setup correctly
bounding_verts = cube_temp_verts.copy()

# Can print off the cube verts
# in order to see the pattern
# print(cube_temp_verts)

# Can otherwise think like a truth table
bounding_verts[0] = [minx,miny,minz]
bounding_verts[1] = [minx,miny,maxz]
bounding_verts[2] = [minx,maxy,minz]
bounding_verts[3] = [minx,maxy,maxz]
bounding_verts[4] = [maxx,miny,minz]
bounding_verts[5] = [maxx,miny,maxz]
bounding_verts[6] = [maxx,maxy,minz]
bounding_verts[7] = [maxx,maxy,maxz]
```

Building Our Mesh

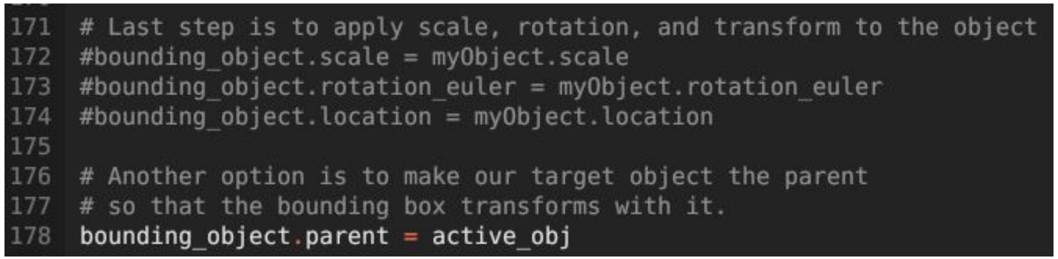
- Finally it's time to create our mesh
  - We'll give it a unique name
  - Populate the mesh from our collection of vertices
    - Importantly using the bounding_verts
    - The edge and face relationship remains the same as a standard cube
- At line 164 and 166, observe that we need to do two steps
  - One to create the object
  - A second step to add it to our scene ('Collection' being the default scene)
- And finally, as an added touch at line 169 -- set the display_type to 'WIRE'
  - (Which I learned by clicking around the user interface)



```
157  # New mesh name
158  bounding_name = "bounding_"+active_obj.name
159  # Create a new 'empty mesh'
160  bounding_mesh= bpy.data.meshes.new(bounding_name)
161  # Populate the mesh with geometry data
162  bounding_mesh.from_pydata(bounding_verts,cube_temp_edges,cube_temp_faces)
163  # Create the new object with a name and associated mesh
164  bounding_object = bpy.data.objects.new(bounding_name, bounding_mesh)
165  # Link in the mesh to a scene so we can actually view it.
166  bpy.data.collections['Collection'].objects.link(bounding_object)
167
168  # Set the bounding box to wireframe by default
169  bpy.data.objects[bounding_name].display_type = 'WIRE'
```

One Final Step

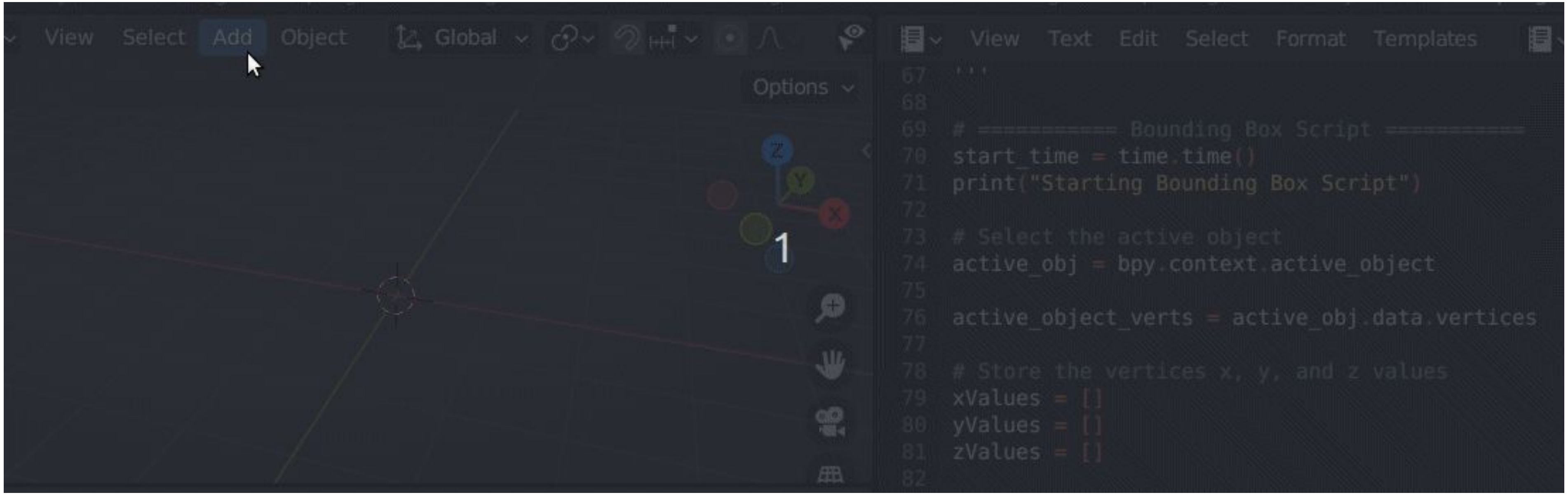
- In order to orient our bounding box to the object, we again have two strategies:
    - Set the transform (scale, rotation, and location) equivalent to the object
    - or
    - Make the bounding object a child of the selected object
        - Thus inheriting the transformations
- Either is fine -- the point is to play around and be creative
    - (Though making the child may be easier to maintain and organize in your scene!)



```
# Last step is to apply scale, rotation, and transform to the object
#bounding_object.scale = myObject.scale
#bounding_object.rotation_euler = myObject.rotation_euler
#bounding_object.location = myObject.location

# Another option is to make our target object the parent
# so that the bounding box transforms with it.
bounding_object.parent = active_obj
```

The Final Result!

- Creating a Bounding Box programmatically in Python

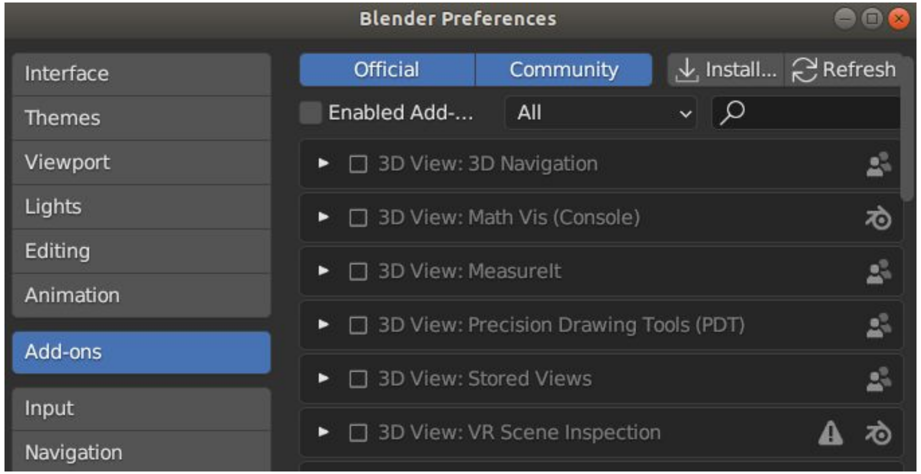


```
'''

# =========== Bounding Box Script ==========
start_time = time.time()
print("Starting Bounding Box Script")

# Select the active object
active_obj = bpy.context.active_object

active_object_verts = active_obj.data.vertices

# Store the vertices x, y, and z values
xValues = []
yValues = []
zValues = []

```

Your Script as an Add-On

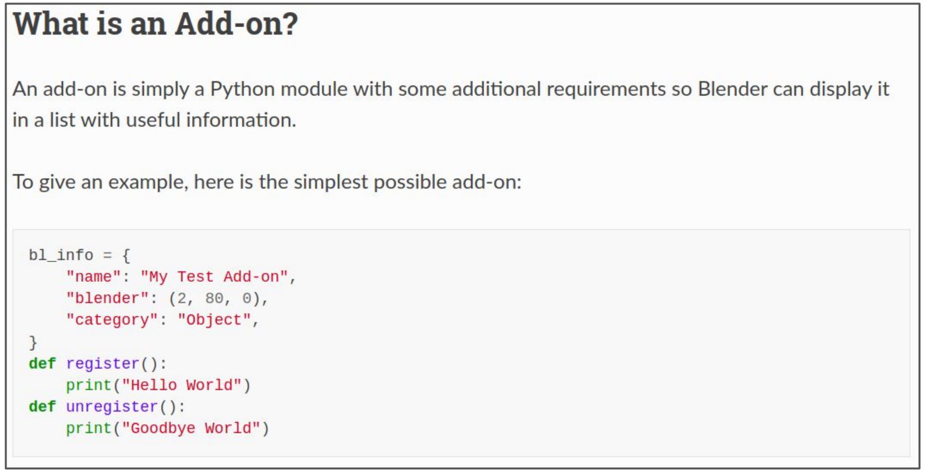
Making our Script available as a Plugin to the World

- Running our script through the Text Editor is perfectly fine
  - However -- it becomes much easier to share and use if we create an ‘add-on’
  - Some add-ons are official, and others are from the community (like you) that we can choose from.



Step 1: Prep

- The first thing we need is to prep our script as an add-on
  - The `bl_info` dictionary populates our plugin with meta-data importantly with:
    - A name
    - Category
  - `register()` and `unregister()` are function calls that take place when we first add our plugin



What is an Add-on?

An add-on is simply a Python module with some additional requirements so Blender can display it in a list with useful information.

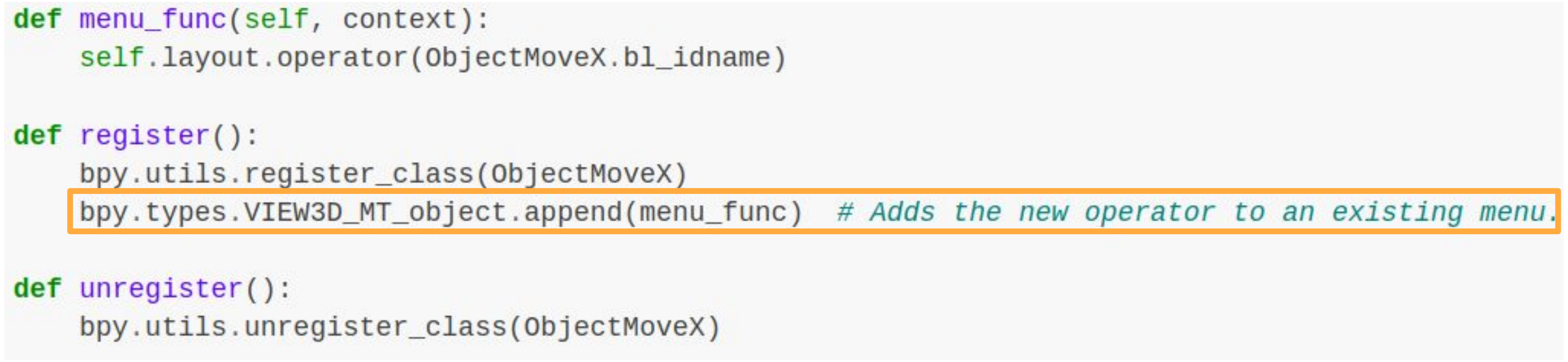
To give an example, here is the simplest possible add-on:

```
bl_info = {
    "name": "My Test Add-on",
    "blender": (2, 80, 0),
    "category": "Object",
}
def register():
    print("Hello World")
def unregister():
    print("Goodbye World")
```

https://docs.blender.org/manual/en/latest/advanced/scripting/addon_tutorial.html

Step 2: Make our command useable

- We can make things slightly more interesting by adding our command to the search (F3) command menu.

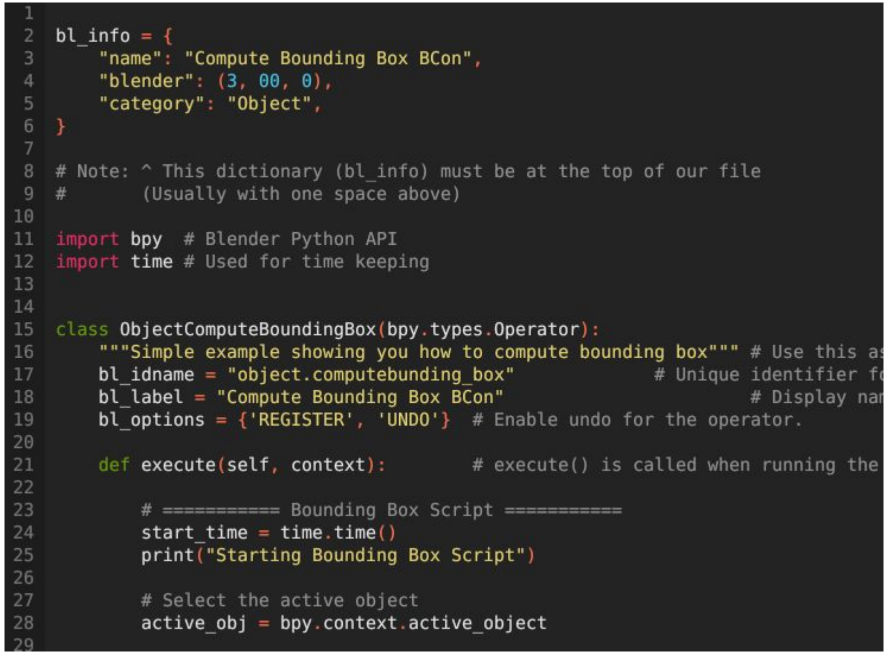


```python
def menu_func(self, context):
    self.layout.operator(ObjectMoveX.bl_idname)

def register():
    bpy.utils.register_class(ObjectMoveX)
    bpy.types.VIEW3D_MT_object.append(menu_func)  # Adds the new operator to an existing menu.

def unregister():
    bpy.utils.unregister_class(ObjectMoveX)
```

https://docs.blender.org/manual/en/latest/advanced/scripting/addon_tutorial.html

Step 3: Prepare ‘execute’ function

- Wrap the work that we previously did into a class
  - This inherits from the ‘Operator’ type, in that we then use our function as an ‘operator’
  - Then... (next slide)



```python

bl_info = {
    "name": "Compute Bounding Box BCon",
    "blender": (3, 00, 0),
    "category": "Object",
}

# Note: ^ This dictionary (bl_info) must be at the top of our file
#        (Usually with one space above)

import bpy  # Blender Python API
import time # Used for time keeping


class ObjectComputeBoundingBox(bpy.types.Operator):
    """Simple example showing you how to compute bounding box""" # Use this as
    bl_idname = "object.computebunding_box"              # Unique identifier fo
    bl_label = "Compute Bounding Box BCon"                        # Display nam
    bl_options = {'REGISTER', 'UNDO'}  # Enable undo for the operator.

    def execute(self, context):        # execute() is called when running the

        # =========== Bounding Box Script ===========
        start_time = time.time()
        print("Starting Bounding Box Script")

        # Select the active object
        active_obj = bpy.context.active_object

```

https://docs.blender.org/manual/en/latest/advanced/scripting/addon_tutorial.html

Step 4: Try it out!

- Test it out -- and we're done!
  - Of course -- for another tutorial we can create a menubar icon and further continue our adventure...maybe next year?

https://docs.blender.org/manual/en/latest/advanced/scripting/addon_tutorial.html

Wrapping Up

Summary

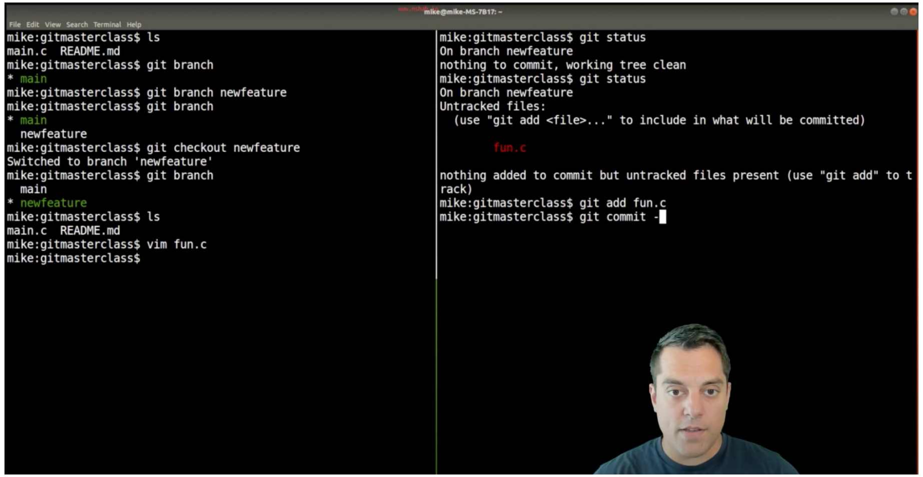
- Today we took an introductory look at Blender 3D's Python API
    - We briefly looked at some of the main modules
    - We solved (or resolved) a non-trivial problem in creating a bounding box
        - We showed how to create an add-on from this script.

Homework: New Feature Ideas of our Script

- What happens if we add new geometry to our mesh?
  - We need a way to poll and recompute the bounding box
  - Investigate handler callbacks here: https://docs.blender.org/api/current/bpy.app.handlers.html
- How about adding an option to creating a bounding sphere?
  - Just need to compute the maximum of the bounds on each axis to use as a diameter.
- Abstraction
  - As an exercise -- think about which chunks of code could go into their own functions
  - Perhaps we could encapsulate this into a classes or files
    - As our scripts get larger, it's important to get a little bit organized.
- Resiliency
  - Add some try/except blocks where necessary to make the code a bit more resilient

Other Essential Skills - Version Control for Text-based Files

- If you’re diving into more programming, version control of your scripts becomes important
  - I’d recommend using ‘git’ and ‘github’ (to host the git repository) as a general skill
    - Git Beginner Masterclass (Free)
  - If folks are already using a tool like ‘perforce’ to manage art assets, that will also work fine too.



```
mike:gitmasterclass$ ls
main.c  README.md
mike:gitmasterclass$ git branch
* main
mike:gitmasterclass$ git branch newfeature
mike:gitmasterclass$ git branch
* main
  newfeature
mike:gitmasterclass$ git checkout newfeature
Switched to branch 'newfeature'
mike:gitmasterclass$ git branch
  main
* newfeature
mike:gitmasterclass$ ls
main.c  README.md
mike:gitmasterclass$ vim fun.c
mike:gitmasterclass$
```

```
mike:gitmasterclass$ git status
On branch newfeature
nothing to commit, working tree clean
mike:gitmasterclass$ git status
On branch newfeature
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        fun.c

nothing added to commit but untracked files present (use "git add" to t
rack)
mike:gitmasterclass$ git add fun.c
mike:gitmasterclass$ git commit -
```

Further resources and training materials

- Best Practices
    - https://docs.blender.org/api/current/info_best_practice.html
    - Goto resource for questions on code structure, performance recommendations, etc.

Random Useful Ideas (If Time) (1/2)

- Check the blender version release
    - # Might be useful for checking compatibility with some feature
    - `import bpy`
    - `bpy.app.version`
        - `or`
    - `major,minor,micro = bpy.app.version`
    - `print(major)`

Random Useful Ideas (If Time) (2/2)

```
# Import our main module
import bpy


# A custom handler that runs only for the ‘Cube’
def my_handler(scene):
    if bpy.context.active_object.name == "Cube":
        print("Cube changed", scene.frame_current)


# ‘Install’ the handler (i.e. function) that will
# run when we do something interesting.
bpy.app.handlers.depsgraph_update_post.append(my_handler)
```

Thank you Tufts SIGGRAPH!

Getting Started with Scripting in Python

with Mike Shah

13:30 - 14:30 Sun, Feb 11, 2024

50 minutes | Introductory Audience

Social: @MichaelShah
Web: mshah.io
Courses: courses.mshah.io
YouTube:
www.youtube.com/c/MikeShah

Thank you!

Extra