The Blender Python API

Precision 3D Modeling and Add-on Development

Chris Conlan

Apress®

The Blender Python API: Precision 3D Modeling and Add-on Development

Chris Conlan
Bethesda, Maryland
USA

ISBN-13 (pbk): 978-1-4842-2801-2 ISBN-13 (electronic): 978-1-4842-2802-9
DOI 10.1007/978-1-4842-2802-9

Library of Congress Control Number: 2017944928

Cover image designed by Freepik

Managing Director: Welmoed Spahr
Editorial Director: Todd Green
Acquisitions Editor: Susan McDermott
Development Editor: Laura Berendson
Technical Reviewer: Justin Mancusi
Coordinating Editor: Rita Fernando
Copy Editor: Kezia Endsley
Compositor: SPi Global
Indexer: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail rights@apress.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/9781484228012. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

For my teachers and colleagues at the University of Virginia Department of Statistics.

Contents at a Glance

Contents

About the Author

Chris Conlan began his career as an independent data scientist specializing in trading algorithms. He obtained his degree in statistics from the University of Virginia where he established himself as an expert in automated trading. His passion for intuitive data visualization introduced him to various 3D modeling and virtual reality suites that he hopes to better integrate into the lives of data scientists. He is currently managing development of private technology companies in high-frequency Forex, machine vision, and precision 3D modeling.

About the Technical Reviewer

Justin Mancusi attended the University of Virginia, where he obtained degrees in computer science and mathematics. In the past, he has worked as an independent consultant at the intersection of computing and statistics. He is experienced in a breadth of computational topics including advanced optimization, computational statistics, and stochastic processes.

Introduction

This text details the development and use of 3D modeling tools in Blender's Python API. We challenge the perception of Blender as purely an artist's tool by building precise data-driven models. Simultaneously, we teach you how aid and enable artists by deploying custom tools in the familiar Blender environment.

The knowledge presented in this text is the result of a deep understanding of not only Blender's documentation and source code, but also of the source code of add-ons written by Blender's core developers. The author has discovered many useful functionalities that are, as of the time of writing, undocumented. Thankfully, we as users can stay on the cutting edge by listening to and learning from those developers. This text unifies well-documented introductory material and undocumented advanced material to create a powerful reference.

This book is packed with code examples and screenshots of powerful scripts and add-ons. We include scripts to automate precise tasks that would otherwise be very difficult to implement by hand. In addition, we build add-ons that augment Blender's existing functionalities with new tools, objects, and customization options.

Definitions

3D modeling is the art of manipulating data to create 3D representations of objects and environments. 3D artists use the following tools and techniques to build 3D models.

- *Manual modeling* involves the artist interacting with a software interface. This can be:
  - Using a 3D modeling suite (Blender, Maya, or 3ds Max) to create and edit objects by hand
  - Playing video games with 3D building elements (Minecraft, Fallout 4, or Sims)
  - Manually inputting data into a 3D object file (.obj, .stl, or .glTF)
- *Automated Modeling* involves algorithmically generating 3D models. This can be:
  - Procedural generation of environments and characters in video games
  - Generating detailed models of buildings from architectural specifications
  - Producing 3D-printed art from fractal algorithms

- *Primitives* are the basic building blocks of 3D models. Though there are no strict rules on what constitutes a primitive, these can be:
  - Simple closed shapes like planes, cubes, and pyramids
  - Simple curved shapes like spheres, cylinders, and cones
  - Complex shapes like tori (plural of torus), Bezier curves, Nurbs surfaces

3D models are data representations of objects and environments. 3D models have the following components.

- *Data formats* allow models to differentiate and specialize by application. Every type of 3D model has a format by which it is specified. These include:
  - Suite-specific formats like .blend for Blender, .3ds for 3ds Max, and .ma for Maya
  - Renderer-specific formats like .babylon for BabylonJS, .json geometry descriptor for 3JS, and .glsl for OpenGL shaders
  - Minimalistic interchange formats like .obj and .stl
- *Vertices and faces* define the points and the surfaces connecting those points in 3D space.
  - Vertices are triplets of real numbers 3D space, or traditional (x, y, z) coordinates of each point of the object.
  - Faces are triplets of integers, where (i, j, k) represents the triangle in 3D space formed by the i-th, j-th, and k-th vertex.

Prerequisite Knowledge for This Book

This book covers Blender version 2.78c running Python 3.5.2. Most examples run on Blender 2.70 and greater, and the concepts apply to Blender generally. Nonetheless, it is recommended that readers use Blender 2.78c to best follow along. As we discuss the history and development of Blender and the Python language, we will point out programming practices that are not likely to work on past and future versions.

We assume a basic working knowledge of Blender and Python 3. Familiarity with any version of Blender 2.60 or greater is sufficient. Similarly, pure Python 2 programmers will have no problem following along.

Material Overview

This text introduces knowledge and sequentially builds on it to create more and more complete and complex software solutions. We introduce and discuss the following major topics.

Chapter 1: The Blender Interface

There are many individual interfaces that make up Blender. The core interfaces are highly scriptable because almost every possible user interaction is tied directly to a Python function. We establish some familiarity with those parts of the interface especially important for Python programming.

The Blender interface will act as both the deployment and development environment for your software. We discuss unique considerations for programming and testing Python while remaining in the Blender interface.

In an effort to minimize usage of screenshots throughout this text, we introduce important vocabulary for discussing the Blender interface. Using this vocabulary, we can focus on Python code while allowing users to work in their own preferred layout of the Blender interface.

Chapter 2: The bpy Module

The `bpy` module is the core of the Blender Python API. Learning to navigate this module will drastically improve your understanding both Blender and the API. Early in this book, we focus on classes within `bpy` that construct objects and manipulate their associated metadata. Later in the book, we access new classes in the `bpy` module that turn scripts into plugins.

The module itself is very verbose. Early scripts will appear both complicated and repetitive. After getting our feet wet with object creation and manipulation, we will begin adding useful function to a toolkit we will build throughout the book. We will store complex and commonly-used algorithms in the toolkit but encourage readers to commit core elements of the `bpy` module to memory. In this way, we create code that is both easy to write and easy to share.

Chapter 3: The bmesh Module

The `bmesh` module is a relatively new module that attempts to simplify complex vertex-level manipulation of object data. For those readers familiar with Blender, most of the operations in `bmesh` will only run in Edit Mode and not Object Mode. This serves to enforce that the functions in `bmesh` are for granular changes rather than global transformation of the mesh data.

This module, in the author's opinion, is what distinguishes the Blender Python API from other automated 3D modeling software. The `bmesh` module gives us algorithmic access to Blender's large suite of Edit Mode tools for vertex-level, edge-level, and face-level object manipulation. It allows us to write procedural generation algorithms for very complex objects in hundreds instead of thousands of lines of code.

Chapter 4: Topics in Modeling and Rendering

It is essential to anyone working in 3D modeling to have a basic understanding of the mechanisms we rely on to render and visualize our work product. We will discuss the basics of rendering pipelines and important rendering topics for Blender Python development. Many perceived bugs and strange behaviors in Blender and in visualizers to which we export are actually intended behaviors of renderers. We learn to detect and program around these behaviors to ensure we are creating highly portable models.

We discuss common and uncommon file formats, Z-fighting, normal vectors, the differences between software and hardware rendering, and much more. This will help us debug Python code based on behaviors we see in various rendering software.

Chapter 5: Introduction to Add-On Development

Bridging the gap between a script and a distributable add-on can be a difficult process that relies on very specific development practices, careful code organization, and occasional meta-programming. Many of these concepts mirror standard Python module development practices, while many others rely on unique behaviors of Blender's scripting interface.

We discuss GUI development, custom Blender data objects, `bpy.types`, and `bpy.utils` in detail here. We discuss organization of add-on files and ways to increase portability across different versions of Blender. At this point in the text, readers will be able to create add-ons that extend Blender to the benefit of modelers that do have Python experience.

Chapter 6: The bgl and blf Modules

The `bgl` module is an OpenGL wrapper for Blender that is useful for marking up, measuring, and visualizing objects and data in the Blender interface. The `blf` module is for drawing text and fonts with the Blender interface and is rarely used without the `bgl` module. We touch on the `bpy_extras` and `mathutils` modules to aid us here.

These modules are incredibly useful for add-on development, because we can influence the data the user sees without affecting the models themselves. We introduce them at this point in the text because their effectiveness depends on the ability to run them as add-ons.

Chapter 7: Advanced Add-On Development

Up to this point, we will have used Blender's Text Editor to create scripts and add-ons. The Text Editor introduces various limitations on the form of our add-ons that we overcome here. We also discuss best practices for data storage and module management by citing popular community add-ons. We conclude this chapter with a discussion of advanced GUI development.

Chapter 8: Textures and Rendering

Up to this point, we will have worked purely with meshes in Blender. In this chapter, we bring scenes to life with texturing and rendering. We discuss procedural *uv*-mapping, lighting placement, and camera positioning. With this discussion comes an overview of lighting types, camera perspective dynamics, and bounding box algorithms.

We conclude this chapter by procedurally rendering an arbitrary scene and providing a framework for automated rendering pipelines. We focus on still renderings in this chapter, but readers interested in automated animation will be able to extend the examples without difficulty.

History of Blender and Python

The relationship between the Blender interface and the Blender Python API is a rare one in the world of software development. It is typical for API-enabled platforms to treat users and developers as separate classes of citizens, complete with separate tools, separate environments, and separate goals. Blender, on the other hand, has erased the line between developers and users, making it easy for users to act as developers and vice versa.

The close relationship between developers and users is the product of wise early design decisions within Blender's core development team. Before Blender was released as free open source software in August 2003 as version 2.26, the core development team released the Python API documentation for the then-premium version 2.25. Python 2.0 had just been released in October 2000, and Blender was already using it to manage calls from the interface to its C-level data structures.

Released in 2009, Blender 2.50 and forward would use pure Python to dispatch editing tasks to its lower-level algorithms and data structures. Every action on the user interface was linked to a Python function, and the user had the option of accessing and calling these functions from consoles and scripts.

As we moved through the early 2010s, Blender artists would become increasingly aware of the influence Python scripting had on the modeling experience. Certain add-ons would become "must-haves" for artists with interests in certain fields. Developers of other 3D modeling software were jumping on the opportunity to develop exporters to port Blender to their software. Today, Blender has its modularity to thank for its massive talent pool, well-paying career opportunities, and active development community.