BlenderPanda Documentation

Release 0.1.0

Oct 25, 2017

Contents

Contents:

CHAPTER 1

Getting Started

Installing BlenderPanda

Dependencies

In order to use BlenderPanda, you need to satisfy two dependencies:

- Blender (tested with 2.77)
- Python with Panda3D SDK (Python 2 and Python 3 are both supported)

Note: The Windows Panda3D SDK installer ships with its own version of Python. This makes a separate Python install uncessary.

The Python binary that has the Panda SDK installed needs to be on the PATH for BlenderPanda to work. On Windows this will likely be the `ppython.exe` that ships with the Panda3D SDK.

Installing the Addon

After dependencies are downloaded and setup, it is time to install the addon itself. There are two ways to get the addon:

- Grab the latest version with Git
- Download a release build

Using Git is recommended to quickly get new updates, but using a release build can be easier to install (especially for those not familiar with Git).

Using Git

GitHub's Download ZIP option does not support git submodules, which are used by BlenderPanda to bring in the BlenderRealtimeEngineAddon. Therefore git the recommended way to grab the latest version of the addon. From the user addons directory (e.g., ~/.config/blender/2.xx/scripts/addons on Linux) use the following git command:

```
git clone --recursive https://github.com/Moguri/BlenderPanda.git
```

To update to the latest version of the addon run the following from the addon's directory:

```
git pull
git submodule update --init --recursive
```

With the addon repository cloned, the addon should now show up in the addons section of Blender's User Preferences and can be enabled from there. If all has gone well, a Panda3D RenderEngine should now be available. If the addon has been enabled, but does not show up in the RenderEngine drop down, check the console for errors.

The mostly likely source of errors is not having Panda3D setup correctly. If, instead, there is an error about not being able to find brte or a submodule in brte, the git repository is likely missing its submodules. This can happen if the `--recursive` option was not used. The following git command should bring in the missing submodule(s):

```
git submodule update --init --recursive
```

Using a release build

Release builds can be found on the project's Releases Page. The Source Code builds will not contain submodules, so make sure you grab the file that starts with `BlenderPanda-v`. Once you have the zip file, the addon can be installed like any Blender addon by following the instructions in the Blender Manual.

Viewport Preview

BlenderPanda is implemented as a Render Engine, so make sure it is selected from the Render Engine dropdown in Blender's info bar. To preview a scene in Panda3D, simply switch to a rendered viewport. Depending on the size of the Blender scene, it may take some time to convert to Panda3D.

Note: Rendered Viewport is not currently supported on macOS

Setting up a Project

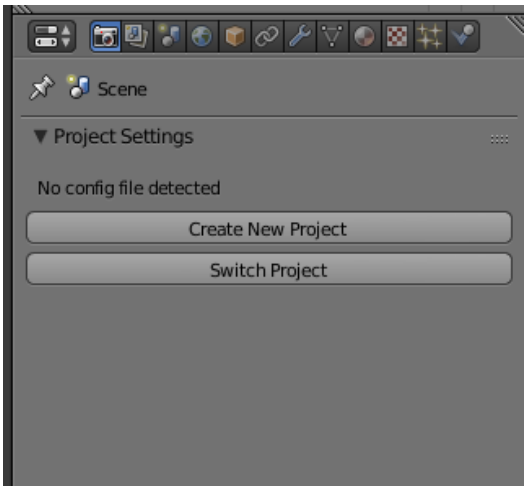
While some features such as viewport previewing and BAM export are possible without a project, to make full use of BlenderPanda you'll need to set one up.

Your two options are to:

1. Start a new game/project using BlenderPanda
2. Setup a BlenderPanda project from your existing game project

Creating a New Project

To create a new project from scratch, click on the Create New Project button in the render properties.



This brings up Blender's file dialog allowing you to select a folder to create the new project in. This will generate a handful of files and directories to get you setup quickly. Assuming a directory called `new_project` was select, the following directory structure is created:

```
new_project/
- assets
- game
|   - blenderpanda
|   |   - bpbase.py
|   |   - __init__.py
|   |   - pman_build.py
|   |   - pman.py
|   |   - rendermanager.py
|   - main.py
- .pman
```

A quick explanation of some of the files:

.pman An INI file that contains settings for your project. This file in the root of your project directory is also how BlenderPanda detects a project. BlenderPanda provides a GUI front-end to the settings in this file.

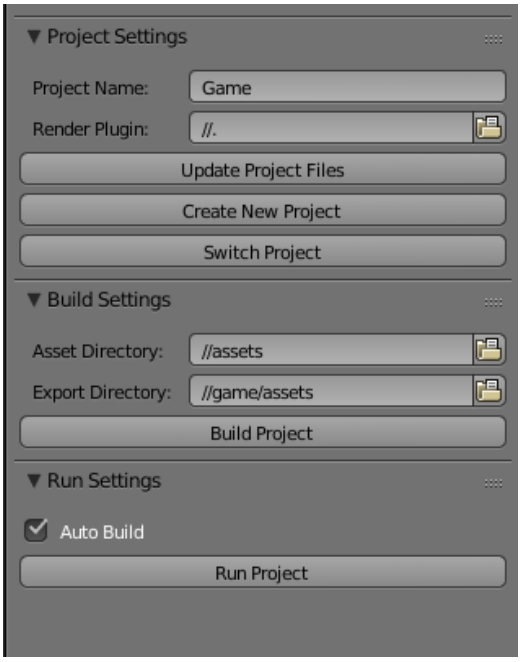
assets The default directory to place source assets (e.g., blend files, images, etc.) to be converted. The location of the assets directory can be modified in `.pman`. **Files must be placed in this directory to be converted by BlenderPanda.**

game/blenderpanda A module that BlenderPanda uses to hook into your game code to provide features such as auto-building. There should be no need to edit the files in this directory.

game/main.py This is the entry point of your Panda3D application. You can do whatever you want with this file, but for the best experience, you should keep the `blenderpanda` initialization.

game/assets While not initially created, this is the default export directory. When building a project, all converted/built files will be put here. This directory is created automatically as part of the build step and should not be put under version control.

When BlenderPanda detects a project, the render properties are updated:



Building and Running the Project

The project can be run via the main.py as any normal Panda3D application or the Run Project button in the render properties in Blender.

If you have any assets that need to be converted, the Build Project button will convert all items in the asset directory and place them in the export directory (creating it if need be). When building a project, a source file in the asset directory will only be converted if it has a newer timestamp than the converted file in the export directory. In other words, only out-of-date assets are converted.

If auto-building is enabled – which it is by default – then the project will be built every time the game is run. This includes running the game from within Blender via BlenderPanda or running `main.py` normally. The build process is very quick if no assets need to be converted, so auto-building should not hurt iteration times when programming. However, auto-build is very useful when pulling in new/modified source assets from a version control system. Therefore, it is recommended to leave auto-building enabled.

When running a project with no changes made to the initial `main.py` created by BlenderPanda, you will be greeted by a window with a boring gray background. This is because there are no models being loaded in the `main.py`. After creating and saving a model to the project's assets directory, it can be loaded by adding the following to `GameApp.__init__` in `main.py` somewhere after the `blenderpanda.init(self)`:

```
self.model = self.loader.loadModel('name_of_model_file.bam')
self.model.reparentTo(self.render)
```

Projects using newer versions of Panda3D (e.g., 1.10 and newer) may want to use the new API style:

```
self.model = self.loader.load_model('name_of_model_file.bam')
self.model.reparent_to(self.render)
```

By default, the camera for the Panda3D scene will be at the origin, so it is likely that the loaded model will not be visible. For information on controlling the camera in Panda3D, please refer to the Controlling the Camera section of the Panda3D Manual.

CHAPTER 2

Managing Projects

pman

`pman` is a Python module used by BlenderPanda to help manage Panda3D projects. It has no dependencies on Blender's `bpy` nor any Panda3D modules, which means it can be used in custom scripts as well.

Configuration

`pman` has two config files that can be used to configure projects: `.pman` and `.pman.user`. The `.pman` file controls project settings that should affect all users running this project, and it should be submitted to version control. `.pman.user` are per-user settings for a project, and it should not be submitted to version control. These per-user settings can be configuration options such as paths to various programs on a users machine. These INI-style files contain key-value pairs divided into sections. Here is an example `.pman` file:

```
[general]
name = Game
render_plugin = game/bamboo/rendermanager.py

[build]
asset_dir = assets/
export_dir = game/assets/
ignore_exts = blend1, blend2

[run]
main_file = game/main.py
auto_build = True
```

`.pman` Options

general options

name The name of the project. This is not currently used for anything.

render_plugin A path to a python file containing a *render manager*.

build options

asset_dir The directory to look for source files (e.g., blend files) that need to be converted.

export_dir The directory to place all converted assets in. This folder is created during the build step and should not be put under version control.

ignore_patterns A comma separated list of Python fnmatch patterns. Any files in the `asset_dir` matching one or more of these patterns will not be converted.

run options

main_file A path to the Python file that contains the entry point for running the game with Panda3D. This is usually the file setting up and running ShowBase.

auto_build When set to True, the project is built (files are converted) every time the project is run. Only outdated files are converted, which makes building very quick if everything is already converted.

auto_save When set to True, the current open file in Blender is saved when using the RunProject operator.

`.pman.user` Options

blender options

last_path The full path to the last Blender binary that opened the project. This can be used to avoid putting a Blender binary in an environment PATH variable. If Blender is not in the PATH, make sure to open the project in Blender once before trying to run/build the project from the command line.

use_last_path If this option is set to True, `pman` will use the `last_path` config value when it needs to launch Blender (e.g., when doing conversion). If this value is set to False, then `blender` must be on the system PATH.

Render Managers

TODO

CHAPTER 3

Indices and tables

- genindex
- modindex
- search