March 2023 Progress Report

Ingame Tutorials

I've implemented a new system for interactive in-game tutorials:

It's still work-in-progress, but it should hopefully make it much easier to learn how PFM works. So far I have only (mostly) finished two tutorials, but I'm planning on creating one for most of PFM's features. Compared to wiki- or video-tutorials, there are several advantages:

- The in-game tutorials can be localized easily into other languages
- If there are interface or functional changes, the in-game tutorials don't become outdated
- It's a more hands-on experience. It's much easier to learn if you're actively doing something compared to watching a video or reading an article.
- They're a part of PFM, so even if the wiki is ever shut down, they'll still be around.

There will be an introductory tutorial, which will lead to several tutorial series covering UI, render effects, common workflows (like importing SFM sessions), etc. This will be my main priority for the next few weeks, as I think it will make a big difference and it'll be worth the time investment □□.

Animating

Constraints

There's a new constraint system with several different types of constraints for animating:

- **Copy Location/Rotation/Scale**: A set of constraints to copy the position, rotation, or scale of an actor to another actor.
- **Limit Location/Rotation/Scale**: A set of constraints to restrict the position, rotation, or scale of an actor to a certain range or axis.
- **Limit Distance**: A constraint that limits the distance between two actors in the scene, ensuring they remain within a specific range of each other. You can choose to make them stay within a certain radius to each other, exactly at a specific radius, or a minimum distance away from each other.
- **Look-at**: A constraint that causes an actor to face towards a target, orienting itself to always point in its direction.

- **Child-of**: A constraint that links two actors together in a parent-child relationship, causing one actor to follow the other's movements and transformations as if it were a child of the parent object.

I used Blender's constraints as a reference point, but I think they should work largely the same as the ones in SFM as well.

(No IK here, just constraints.)

Any two properties of any two actors can be constrained together, as long as they're transform-types (Vector3/Quaternion/etc.). All constraints have some base properties:

- **influence**: This property determines how much the constraint affects the driven actor. It is a value between 0 and 1, where 0 means no influence and 1 means full influence.
- **driverSpace**: This property specifies the coordinate space in which the driver actor (the actor that controls the constraint) is defined. Possible spaces are world/local/object.
- **drivenObjectSpace**: This property specifies the coordinate space in which the driven actor (the actor that is affected by the constraint) is defined. Possible spaces are world/local/object.
- **orderIndex**: This property determines the order in which multiple constraints are applied to the same object. It is an integer value that can be used to prioritize constraints.

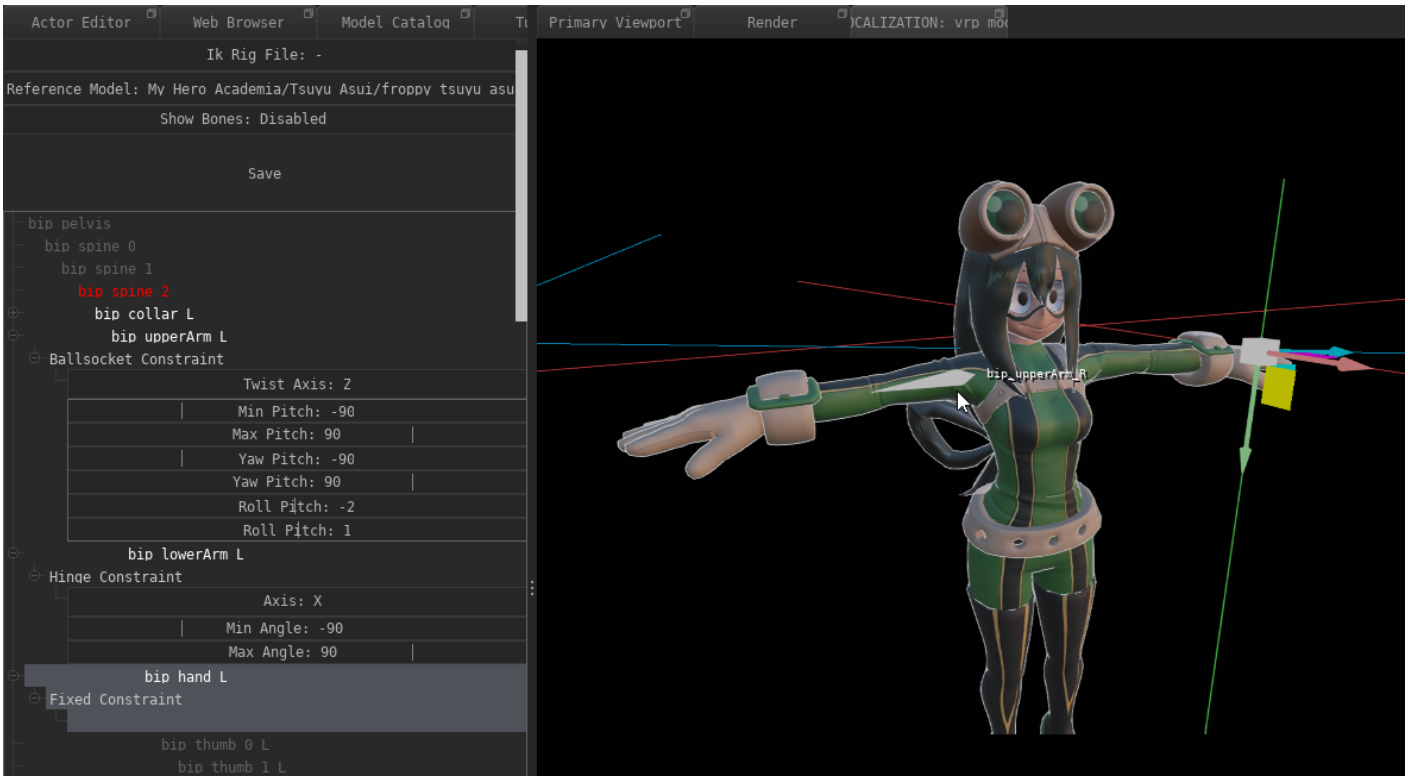
Depending on the constraint type, some additional options are available. Blender also has a "custom space" option for the *driverSpace* and *drivenObjectSpace*, which may come in handy. I haven't added something like that yet, but I may do so.

Inverse Kinematics

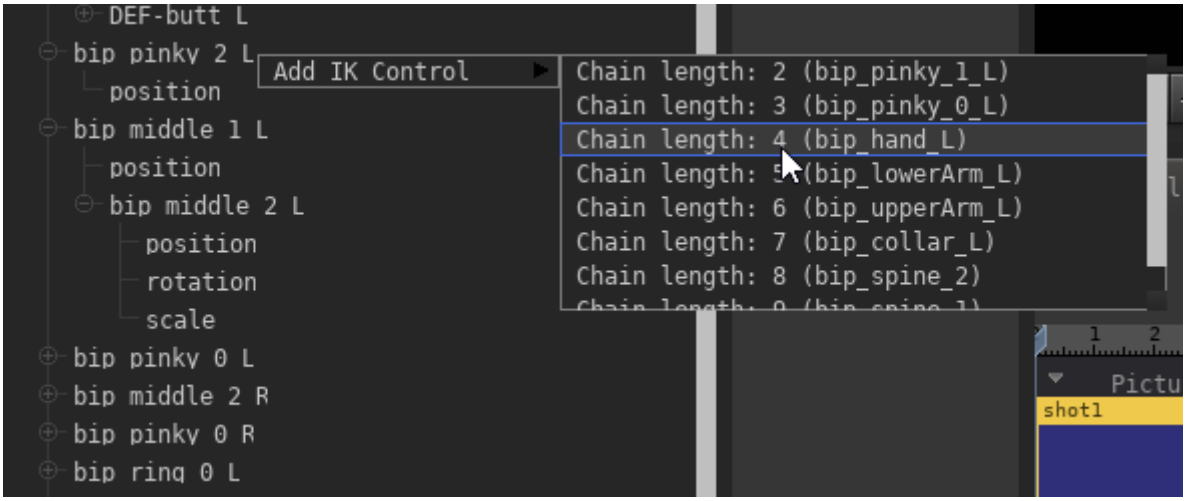
The new (full-body) IK system is now technically fully integrated, however I still have to do a lot of testing and tweaking, so I can't show it off properly yet.
Unfortunately there is also no way for me to add support for SFM IK rigs, since the IK system is a fundamentally different one, and since SFM's IK rigs are python-based, which can't be converted automatically. The good news is I put in a lot of effort to try and simplify the process for creating IK rigs in PFM:

IK Rig Editor

There's a new editor for creating custom IK rigs. The general approach is very different compared to SFM, but hopefully less convoluted since you don't have to mess around with scripting:



The editor is primarily for creating full-body IK rigs, as well as VR Body IK rigs. If you're in a hurry and just want to set up a quick IK chain, there's of course still the option to do so by right-clicking a bone in the actor editor and selecting the desired chain length:



I will showcase these features properly once they're completed.

Improved Lightmaps (again)

Not much to say here, other than that I've once again made some more improvements to the quality of baked lighting:

Before:

lightmap_old.png

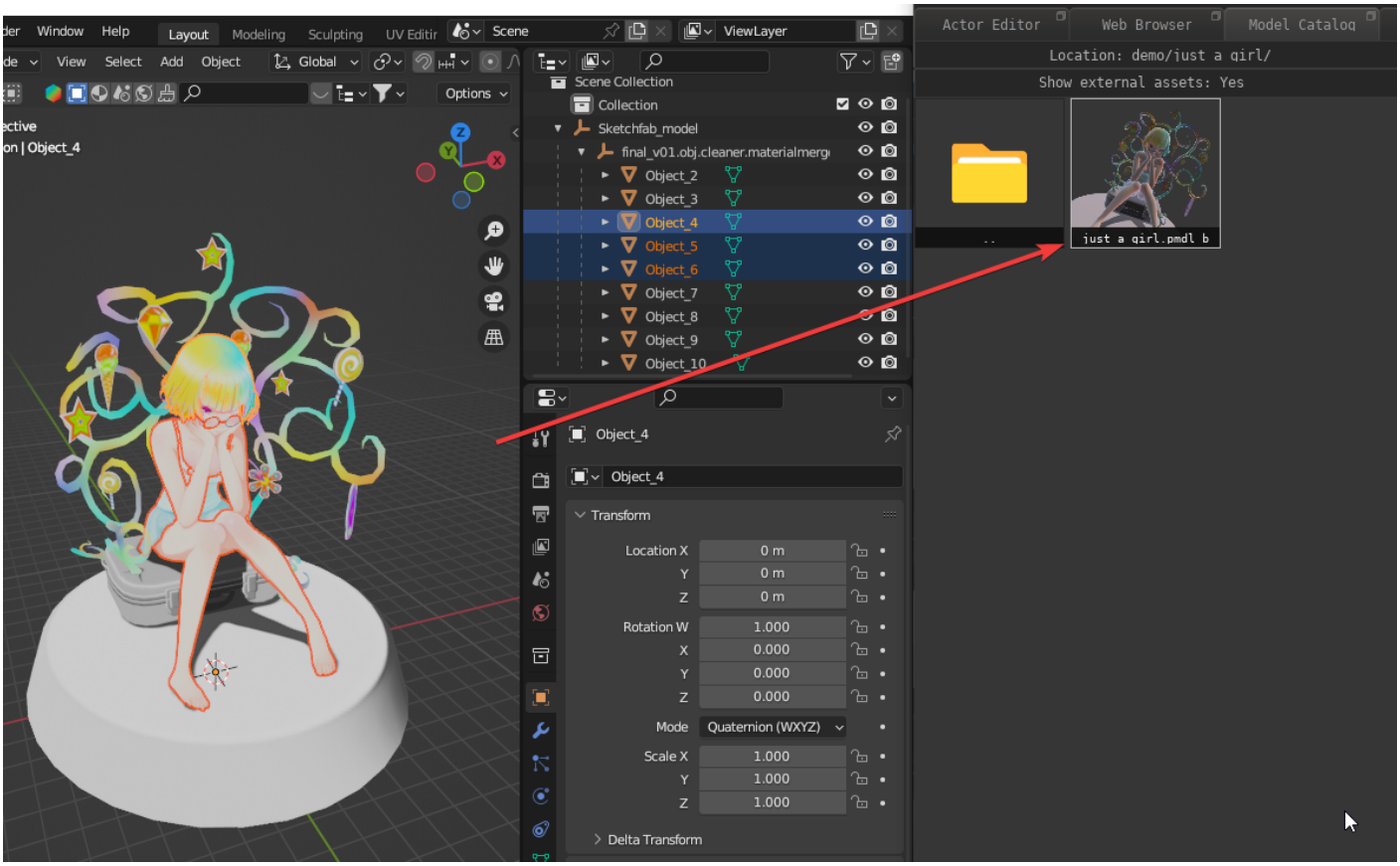
After:

I'm quite happy with the results now. The lighting is distributed more evenly, and I fixed some issues that would cause some areas to appear dark (or completely black), as well as some other minor issues.

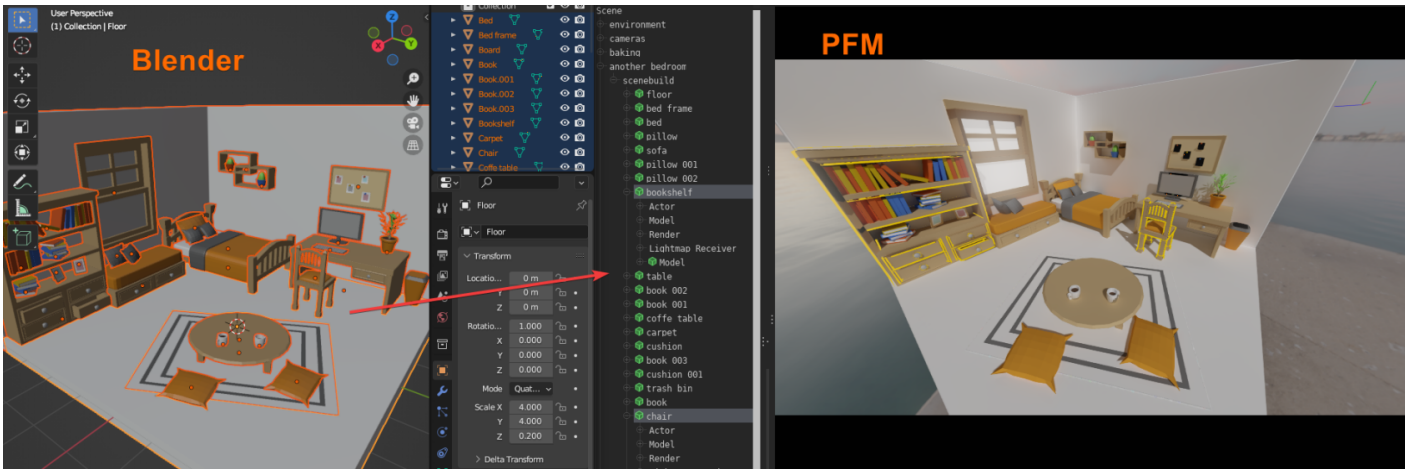
glTF Scene Import

> Reminder: glTF is an asset format which can contain model and scene data (with multiple objects, cameras and light sources). It's the primary format for importing/exporting assets to/from Pragma. (More information here.)

I have made some more improvements to the glTF asset import. Previously importing a glTF asset into Pragma would simply create a single model out of it:



In cases like this where all the parts belong together, or it's just a single character, it makes sense to do this.
For cases where the glTF contains an entire scene, I've now added an alternative import method. With this method, all of the scene objects are imported as individual models, as well as a map containing the objects with their original poses:



(Every object is also imported as a separate model.)

There's also been a bunch of fixes to object scaling and coloring, etc. There's still no way to export an entire PFM scene as glTF (only individual models), but I'll get around to that eventually.

Cel Shading Outline

A minor new feature but still worth mentioning I think. I added a simple cel-shading outline effect which can be added to actors:

It's currently just the outline though, there is no toon shader for characters at the moment.
You can customize the size, color and emissivity of the outline, but it only works when rendering with the Pragma renderer (i.e. not with Cycles).

Tiled Rendering

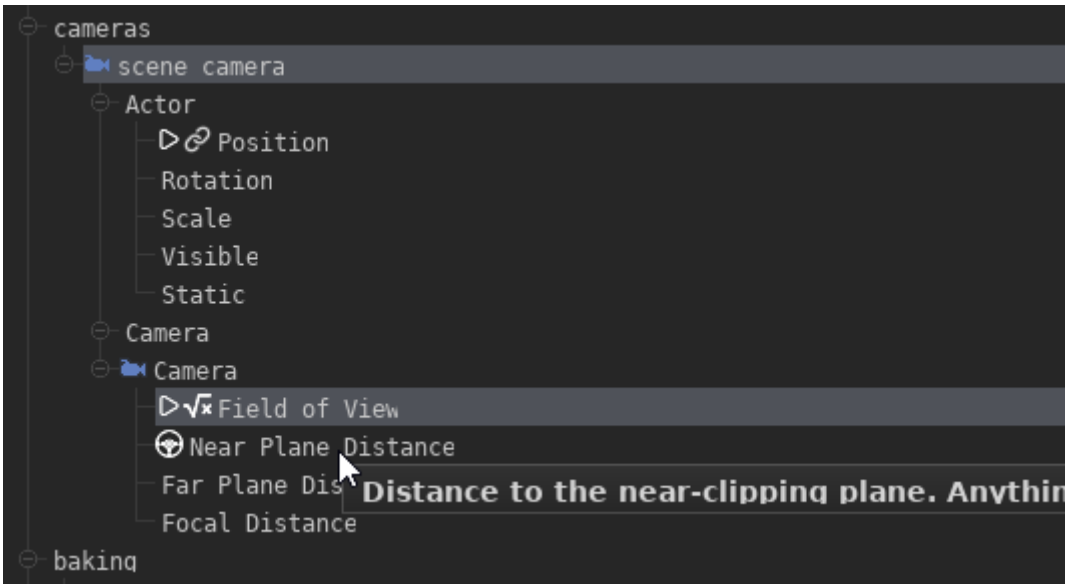
I've added tiled rendering support for the Pragma renderer. This allows you to render images at any resolution with high-quality SSAA, regardless of available VRAM:

https://pragma-engine.com/share/patreon/23-03-30/tiled_rendering_example.png (**Warning:** 165 MiB image)

It's currently not very fast, so I still have to work on some performance improvements before I can publish it with the next update. It also doesn't work for VR renders yet, those are going to need some additional treatment.

Actor Editor Improvements

I've made a bunch of quality-of-life improvements to the actor editor:



- Most properties now have descriptive tooltips
- Components, properties and tooltips are now localized (i.e. available in other languages)
- There are new icons, depending on whether a property is animated, has constraints, a math expression or an animation driver. Clicking on the icon will directly lead to whatever its refering to (e.g. the constraint actor).

Map Scenebuilds

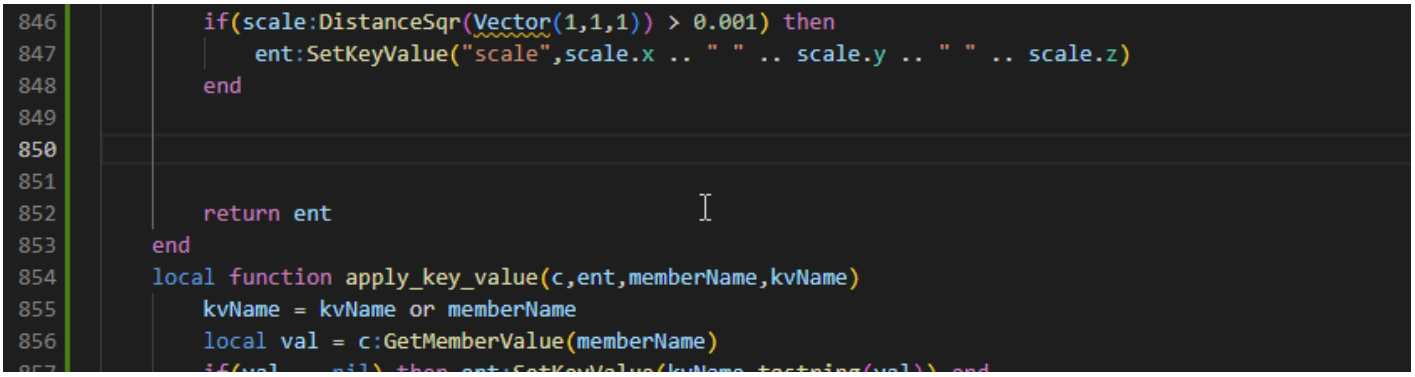
PFM projects can now be exported as maps. This will take all of the static actors (and light sources/baked lighting) from the project and generate a map from it.

Basically the general idea is that you can create a scenebuild in PFM (i.e. a scene with only static props and static lighting and no animated actors) and export it as a map. This allows you to easily re-use the same scenebuild for different projects, or share it with other people, without having to copy the actors from one project to another.

You can also take an existing map, import it into a PFM project, make some changes to it, and then export the result as a new map. This way you could also merge multiple maps into one if you wanted to.

Lua Debugging with Visual Studio Code

I stumbled upon some great Lua extensions for Visual Studio Code. Previously I had already added support for the ZeroBrane Lua IDE, but unfortunately that one is no longer in development, so Visual Studio Code will be the recommended IDE for Lua development in Pragma from now on. I added support for the extensions to Pragma, which enables code auto-completion, step-by-step debugging, conditional breakpoints, etc:



It also works great with GitHub Copilot (the AI-assistant by GitHub), which was actually the main reason I wanted to give it a shot. If nothing else, this will make my life a little easier for sure.

There's a tutorial on how to set up on the wiki, but it's not quite complete yet and won't work until the next update.

Conclusion

Unfortunately the new features are not published yet. Some of the new features are not quite finished yet, I still have to do a bunch of testing, work on performance improvements, bug fixing, code cleanup, etc., so I won't be working on any new features (other than the tutorials) until that's all completed. I'll do my best to get all of that done asap.

I have also purchased a new domain for Prelewd, prelewd.com, in preparation for the next Prelewd release. It's still under construction, but once I got all of the new features and tutorials completed, I do have a bunch of things planned.

Changelog

I don't have time to write a changelog by hand at the moment, but I'm experimenting with generating changelogs automatically.

You can find the full changelog over here, but it may include some older or duplicate features/changes.

Revision #15
Created 2023-03-23 06:29:36 UTC by Silverlan
Updated 2025-01-19 14:13:14 UTC by Silverlan