Assets

Loading Assets

Gym currently supports loading URDF and MJCF file formats. Loading an asset file creates a `GymAsset` object that includes the definiton of all the bodies, collision shapes, visual attachments, joints, and degrees of freedom (DOFs). Soft bodies and particles are also supported with some formats.

When loading an asset, you specify the asset root directory and the asset path relative to the root. This split is necessary because the importers sometimes need to search for external reference files like meshes or materials within the asset directory tree. The asset root directory can be specified as an absolute path or as a path relative to the current working directory. In our Python examples, we load assets like this:

```
asset_root = "../../assets"
asset_file = "urdf/franka_description/robots/franka_panda.urdf"
asset = gym.load_asset(sim, asset_root, asset_file)
```

The `load_asset` method uses the file name extension to determine the asset file format. Supported extensions include **.urdf** for URDF files, and **.xml** for MJCF files.

Sometimes, you may wish to pass extra information to the asset importer. This is accomplished by specifying an optional `AssetOptions` parameter:

```
asset_options = gymapi.AssetOptions()
asset_options.fix_base_link = True
asset_options.armature = 0.01

asset = gym.load_asset(sim, asset_root, asset_file, asset_options)
```

Loading Meshes in Assets

Gym uses Assimp to load meshes specified in assets. Some meshes can have materials or textures specified directly in the mesh file. If the asset file also specifies a material for that mesh, then the materials from the asset take priority. To use the mesh materials instead, use

asset_options.use_mesh_materials = True.

Gym tries to load normals directly from the mesh. If the mesh has incomplete normals, Gym will generate smooth vertex normals. To force Gym to always generate smooth vertex normals/face normals, use

asset_options.mesh_normal_mode = gymapi.COMPUTE_PER_VERTEX

or

asset_options.mesh_normal_mode = gymapi.COMPUTE_PER_FACE

respectively.

If a mesh has submeshes that represent a convex decomposition, Gym can load the submeshes as seperate shapes in the asset. To enable this, use

asset_options.convex_decomposition_from_submeshes = True.

Overriding Inertial Properties

Inertial properties for rigid bodies are important for simulation accuracy and stability. Each rigid body has a center of mass and an inertia tensor. URDF and MJCF allow for specifying these values, but sometimes the values are incorrect. For example, there are many URDF assets in the wild with seemingly arbitrary inertia tensors, which can cause undesirable simulation artifacts. To overcome this, Isaac Gym allows for explicitly overriding the center of mass and inertia tensors using values computed from the geometries of collision shapes:

```
asset_options.override_com = True
asset_options.override_inertia = True
```

By default, these options are both False, which means that the importers will use the values given in the original assets.

See `python/examples/convex_decomposition.py` for sample usage.

Convex Decomposition

Isaac Gym supports automatic convex decomposition of triangle meshes used for collision shapes. This is only needed when using PhysX, since PhysX requires convex meshes for collisions (Flex is able to use triangle meshes directly). Without convex decomposition, each triangle mesh shape is approximated using a single convex hull. This is efficient, but a single convex hull may

not accurately represent the shape of the original triangle mesh. Convex decomposition creates a more accurate shape approximation consisting of multiple convex shapes. This is particularly useful for applications like grasping, where the exact shape of physical objects is important.

By default, convex decomposition is disabled. You can enable it using a flag in the AssetOptions during asset import:

```
asset_options.vhacd_enabled = True
```

Gym uses a third party library (V-HACD) to perform the convex decomposition. There are many parameters that affect the decomposition speed and the quality of the result. They are exposed to Python in the `isaacgym.gymapi.VhacdParams` class, which is included in AssetOptions:

```
asset_options.vhacd_params.resolution = 300000
asset_options.vhacd_params.max_convex_hulls = 10
asset_options.vhacd_params.max_num_vertices_per_ch = 64
```

See `python/examples/convex_decomposition.py` for sample usage.

Convex decomposition can take a long time, depending on the paramaters and the number of meshes. To avoid recomputing convex decompositions every time, Gym uses a simple caching strategy. When a mesh is decomposed for the first time, the results of the decomposition will be stored in a cache directory for fast loading on subsequent runs. By default, the cache directory is `${HOME}/.isaacgym/vhacd`. The cache directory and any files it contains can be safely deleted at any time.

You can view the results of convex decomposition by clicking on the Viewer tab in the viewer GUI and enabling the “Render Collision Meshes” checkbox.

Procedural Assets

Simple geometric assets like boxes, capsules, and spheres can be created procedurally:

```
asset_options = gym.AssetOptions()
asset_options.density = 10.0

box_asset = gym.create_box(sim, width, height, depth, asset_options)
sphere_asset = gym.create_sphere(sim, radius, asset_options)
capsule_asset = gym.create_capsule(sim, radius, length, asset_options)
```

Asset Options

See `isaacgym.gymapi.AssetOptions`.

Asset Introspection

You can inspect the collections of components in each asset, including rigid bodies, joints, and DOFs. See `examples/asset_info.py` for sample usage.

Creating Actors

Loading or creating an asset does not automatically add it to the simulation. A `GymAsset` serves as a blueprint for actors and can be instanced multiple times in a simulation with different poses and individualized properties, as described in the section on Environments and Actors.

If the collision filter for the actor is set to -1, the actor will use filters loaded in by the asset loaders. This is important for MJCF files that specify non-zero contypes/conaffinities or have other contacts specified. Setting the collision filter to 0 will enable collisions between all shapes in the actor. Setting the collision filter to anything > 0 will disable all self collisions.

Limitations

The asset pipeline is a work in progress, so there are some limitations.

- The URDF importer can only load meshes in OBJ format. Many URDF models come with STL collision meshes and DAE visual meshes, but those need to be manually converted to OBJ for the current importer.
- The MJCF importer supports primitive shapes only, such as boxes, capsules, and spheres. Mesh loading is currently not available in that importer.
- The MJCF importer supports multiple joints between a pair of bodies, which is useful to define independently named and controllable degrees of freedom. This is used in the `humanoid_20_5.xml` model to define independent motion limits for shoulders, hips, and other compound joints.
- The MJCF importer only supports files where the worldbody has no more than one direct child body. This means that MJCF files that define an entire environment may not be supported. For example, one MJCF file cannot contain both a robot and a ground plane.

Relevant Examples

Take a look at the Python examples `asset_info.py` and `joint_monkey.py` for working with assets.