The Guide to Ansel

December 5, 2018

1 Getting Started

Contents

1.1 Prerequisites

- Visual Studio 2017
- Windows 10 SDK 10.0.1734

1.2 Installing

Getting Ansel up and running is an easy process.

1. First, go to `www.github.com/maxortner01/ansel`.
2. Download the repository as a `.zip` file.
3. Extract the `.zip` to a folder on your computer.
4. Double-click on the `.sln` in the folder (assuming Visual Studio 2017 is installed)

The Solution should run the Game project without any problems. If problems should arise, be sure to update Visual Studio through the Visual Studio Installer and ensure Universal CRT is installed.

2 Making Your Own Game

2.1 Making the Project

Using Ansel and making your own game project is super easy. The `.zip` contains all of Ansel's source code and everything, but if your intent is only to *use* Ansel rather than develop on it, you don't need most of that. In fact, you can make your own solution without Ansel and it should still run if you follow the next steps.

2.2 Linking Ansel

In order to link Ansel into your project the following steps must be taken.

1. When you have created a Visual Studio Project, right-click on it and select `Properties`.
2. Then go to `Linker` and where it says `Additional Library Directories` input the directory to `bin/'Your Configuration'/Ansel` which is where `Ansel.lib` is located as well as `bin/'Your Configuration'/AnselECS` which is where `AnselECS.lib` is located.
3. Next, go `Linker->Input` and where is says `Additional Dependencies` and click the down arrow.
4. Finally, add `Ansel.lib` and `AnselECS.lib`

2.3 Including Ansel

The final step is to include the source directories for Ansel and the Component System.

1. Right-click on your project and select `Properties->C/C++`.
2. Where it says `Additional Include Directories` click the drop-down and add the folders `Ansel/src` and `AnselECS/src`.

Now Ansel is ready to go!

3 Running the Code

3.1 Entry Point

With Ansel's `include` directory included, the `.lib` file linked, and the `.dll` in the working directory, its time to start coding. Ansel is built to be easy to use and it shows in the coding process.

Ansel works by rendering classes inherited from `Ansel::Screen`. This class contains a pure virtual function (`onUpdate()`) that is to be overridden with code that runs every frame. Before this, though, C++ needs an entry point.

```cpp
#include <Ansel.h>

#include "NewScreen.h"

int main() {
    /* ... Ansel goes here ... */
}
```

In order to run Ansel, three things are needed:

1. An `Ansel::Window` instance.

   ```cpp
   Ansel::Window window(1920, 1080); // Width and height of the window
   ```

2. A user-created `Ansel::Screen` subclass.

   ```cpp
   Game::NewScreen screen(&window);
   ```

3. Finally, a `Ansel::Engine` instance that puts everything together.

   ```cpp
   Ansel::Engine engine(&window, &screen);
   ```

Now, to run the engine, one more line is needed:

```cpp
engine.run();
```

After this, Ansel is off, running everything including `screen.onUpdate(float timeDelta)` every frame.

3.2 Screen Subclass

This code runs Ansel, but there's nothing but a black screen! The code that runs that defines your program goes within the `onUpdate()` function of a `Ansel::Screen` subclass. To do this you must first create a class that defines the screen space which inherits from `Ansel::Screen`. `Ansel::Screen` itself has a constructor that takes a pointer to a `Ansel::Window` instance, so this must be considered when making the subclass's constructor.

```
#include <Ansel.h>

using namespace Ansel;

namespace Game
{
    // NewScreen inherits from Ansel::Screen
    class NewScreen : public Screen
    {
        /* Private game variables go here... */
        unsigned int uFrame = 0;

    public:
        // NewScreen's constructor must also initialize Screen
        NewScreen(:Window* w) : Screen(w) {}

        void onUpdate(float timeDelta) {
            /* All event and rendering goes here... */
            uFrame++;
        }

        void onCreate()  override {
            /* ... initialization code goes here ... */
        }

        void onDestroy() override {
            /* ... destruction code goes here ... */
        }
    };
}
```

The two last methods (`onCreate() and onDestroy()`) are optional and are invoked either when Ansel recognizes everything else is done initializing or being destroyed.

4 Models and RawModels

The two most important data types in Ansel are `Models` and `RawModels`.

`RawModel`	Raw data container. This holds a 3D Model's vertex, normal, texture, material, and color information. There should only be *one* `RawModel` per 3D model.
`Model`	A `Model` acts only as transformation information with a reference to a `RawModel`. There can be thousands of `Models`, each with their own transformation (location, rotation, and scale) information, but all referencing one `RawModel` that is thir graphical representation.

`Models` and `RawModels` can be created through the `Loader`.

```
// You can hold all model pointers yourself, or you can simply just call
// Loader::getRawModel(name) or Loader::getModel(name) to retreive the
// instances
RawModel* objFile = Loader::makeRawModel("path_to_file.obj", "rawModel");

// There are many overloads of makeRawModel, with a ton of different
// combinations of data, here we are defining our own data
std::vector<vec3f> vertices = { ... };
std::vector<vec3f> normals  = { ... };
std::vector<vec4f> colors   = { ... };
RawModel* customModel = Loader::makeRawModel(vertices, normals, "customRawModel");

// If colors.size() is the same as customModel->getVertexSize() then
// all the instances of the model will have the same color
// Otherwise, you can put in a list of colors for every model manually
// but Ansel does this already for you
// By default the color is black
customModel->loadColors(colors);

// Just like eveything before, you can make a model through the
// Loader by passing a pointer to a RawModel
Model* objModel = Loader::makeModel(objFile);
objModel->setLocation({ 1, 0, 1 });
```

5 Making an Entity

Everything that represents something in your game is going to be an Entity. Creating an Entity is easy (and should be done in the `onCreate()` function of your own `Ansel::Screen` class). Keep in mind that if you utilize the naming functionality, you don't have to concern yourself with storing any pointers/entities.

```
void onCreate() override {
    // Create the Entity
    Loader::makeEntity("cube");

    // Get the entity instance
    ECS::EntityInstance entity = Loader::getEntity("cube");
}
```

> *While I prefer the pointer notation (`Entity*`), I have provided a less verbose alternative which is the addition of `Instance` after a classname (which is the same as a pointer).*

As you can see, we create the entity with the `Loader` and give it a name, in this case its `cube`. Now, in order to get a local variable we can get the Entity with another call to `Loader`, this time invoking `getEntity`.

5.1 Adding and Making Components

At the moment there is one important component inherit to Ansel, and that is a `Model`. `Models` are derivatives of `ECS::Renderable` and therefore are passed to the Renderer. In order to add a `Model` to an entity you can do the following.

```
RawModel* rawModel = Loader::makeRawModel("path_to_obj.obj", "rawModel");
Model*    model    = Loader::makeModel(rawModel, "model");

entity->addComponent(model, "model");
```

Now, if you wish to manipulate the model. You must cast it back into the original type.

```
ECS::EntityInstance entity = Loader::getEntity("cube");
Model* model = entity->getComponent("model")->cast<Model*>();
```

Now you can use model and manipulate model.

6 Troubleshooting

- A lot of errors on compile with descriptions like `"cannot open source file errno.h"` or `"the global scope has no acosf"`

Make sure Windows 10 SDK 10.0.1734 is installed.

- Low FPS

  The reason for low FPS could simply be unoptimized code, out-of-date drivers, or an older graphics card. However, another problem could be OpenGL simply not using the GPU. With Nvidia cards, to fix this, open up the Nvidia control panel. Go to `Manage 3D settings` and set "Preferred Graphics Processor" to "High-performance NVIDIA processor."