With apologies, the build process here is pretty crude. In particular, I haven't figured out how to use cmake with ISPC, so the directories that build ISPC stuff get built using a Makefile, which you'll have to edit to point to appropriate places for OSPRay and ISPC. **NOTE: when you build OSPRay, you will need to set OSPRAY_BUILD_ISA appropriately - on Maverick, 'avx' works.**

## 1. Build ispc VisRenderer

This is a customization of the OSPRay RaycastVolumeRenderer to handle implicit slicing planes and multiple light sources. Hopefully will eventually show up in OSPRay. Note that this uses a simple Makefile and builds in place.

cd VolumeViewer/src/ispc

- edit the Makefile to point to the OSPRay source and ISPC compiler

make

Should create libvis.so

## 2. Build common library

These are various classes to handle common functionality between the Viewer and Cinema versions.

cd VolumeViewer/sr
mkdir common.obj

Note the name of this object directory - other stuff will depend on it, so don't get creative here

cd common.obj
cmake -DOSPRAY_SRCDIR=*wherever* ../common

You'll need to set OSPRAY_SRCDIR to point to the OSPRay source

make

should create libcommon.so

## 3. Build the viewer

This is a GUI for interactive viewing of volumes.

cd VolumeViewer/src
mkdir viewer.obj

Note the name of this object directory - other stuff will depend on it, so don't get creative here

cd common.obj
cmake -DOSPRAY_SRCDIR=*wherever* -DOSPRAY_OBJDIR=*wherever* ../viewer

You'll need to set OSPRAY_SRCDIR to point to the OSPRay source and OSPRAY_OBJDIR to point to wherever you built OSPRay. You'll need Qt for this.
j
make

should create

## 4. Build the Cinema library and test executable

This is the library used to instrument a simulation and create Cinema databases.

cd VolumeViewer/src
mkdir viewer.obj

Note the name of this object directory - other stuff will depenjkjd on it, so don't get creative here

cd common.obj
cmake -DOSPRAY_SRCDIR=*wherever* -DOSPRAY_OBJDIR=*wherever* ../cinema

You'll need to set OSPRAY_SRCDIR to point to the OSPRay source and OSPRAY_OBJDIR to point to wherever you built OSPRay.

make

## 5. Build the Perlin simulation simulation.

This code evolves a 3D noise function through time to simulate a simulation. A library - libperlin.so implements the noise function (using ISPC). Two executables are included: **dump**, which can be used to write a test timestep, and **sim**, which runs through a set of time steps and uses the Cinema library to generate a Cinema database.

cd VolumeViewer/src/perlin

- edit the Makefile to point to the OSPRay source and ISPC compiler

make

## 6. Set various environment variables

Sets PATH, LD_LIBRARY_PATH and COLORMAP_DIR - which is where to find the colormaps

cd VolViewer

-- edit **vv.env** to set **VOLVIEWER_ROOT** to point to the directory containing the VolViewer code and **OSPRAY_RELEASE** to point tot he directory where you built OSPRay.

. vv.env

Note - you'll have to twiddle this if you use anything other than bash.

## 7. Create test dataset

In this step we will (try to) use the Perlin **dump** program to create a test dataset

cd VolumeViewer
mkdir test
cd test
dump -r 256 256 256

This should create two files: **data.raw** containing 256x256x256 flowting point numbers and **data.vol** to describe it.

## 8. View the test dataset

viewer
File->Open
select **data.vol**

You might need to mouse-down and drag in the window to get it to show up. Now you can use the slicing planes to clip and optionally be visible, and enable and set isovalues. To see anything inside the volume you'll need to move the slider next to the transfer function way down.

File->Save State
save as **test.state**

Now exit and restart:

viewer test.state

You'll be back to where you were. Note that isosurfaces might not show up until you tweak the appropriate sliders.

## 9. Test the Cinema library

For this we use **test_cinema**, for which source is in the **cinema** source directory. This uses the single timestep dataset built earlier. By default, this is generates images varying for a 3 values of phi, 3 isovalues and 12 values of theta, for 108

total images, with ancillary data files and info.json making up a Cinema database.

test_cinema -s 512 512 test.state

**10. Run the simulation simulation**
Now we do the same thing using the perlin noise simulation. **Note: this will not recreate or overwrite older files of the same name.** This is a *feature* so the thing can be restarted.

rm *.png *.__data__ info.json
sim -s 512 512 -r 256 256 256 -t 0.05 20 test.stat

I forget exactly what options are set here, but this will run through 20 time steps, creating a Cinema database. This will generate a *lot* of frames.