DRAFT - Datacube Rendering with Blender

Dr. Brian R. Kent, Scientist, National Radio Astronomy Observatory
Twitter, Instagram, YouTube: @VizAstro
https://www.cv.nrao.edu/~bkent/blender

Visualization Tutorial

October 2021 Flatiron Institute

Requirements: Linux, Windows, or Mac computer

A laptop is sufficient, but a two button mouse with a wheel is preferred to a trackpad.

If you have a full keyboard with a right-side numeric keypad, it makes navigation easier.

Blender download (https://www.blender.org/)

Data from CT scans:: http://graphics.stanford.edu/data/voldata/

Any sliced datacube data will work

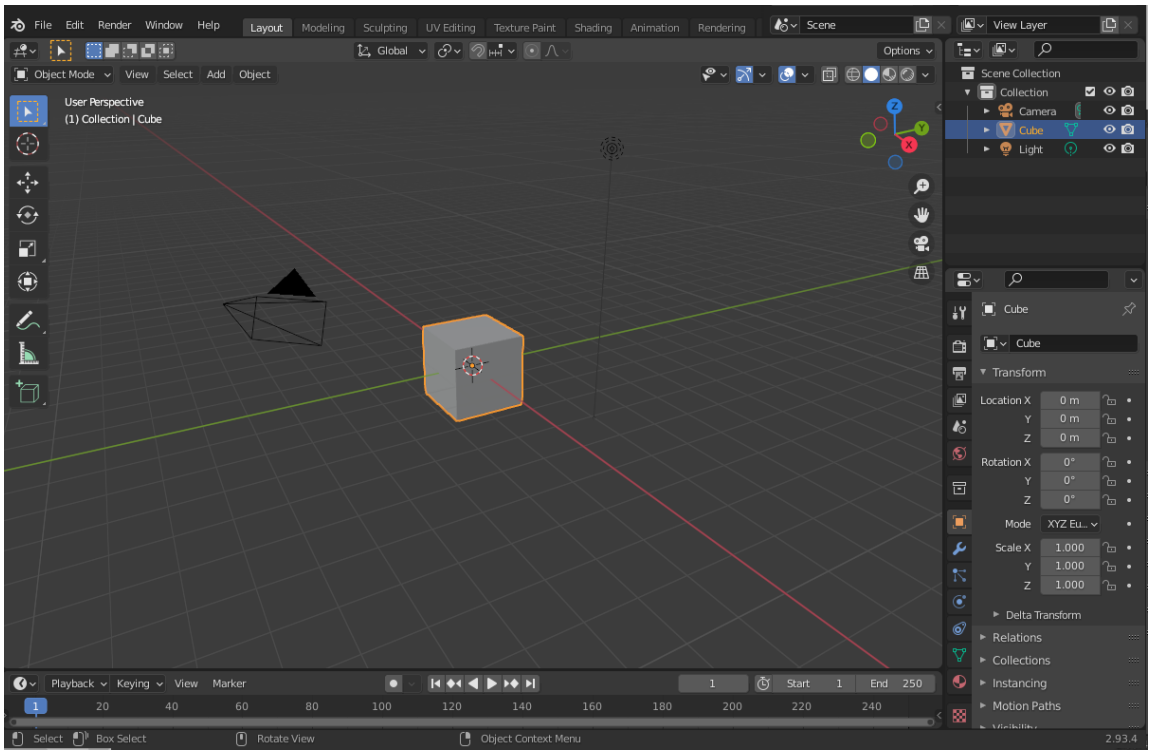
Basic knowledge of Python and a bit of Open Shading Language C will be used.

Git repository: **git clone https://github.com/brkent/flatiron2021.git**

We will render a 3D data cube as a transparent volume. This tutorial will use the following concepts:

- Working with the Blender interface, keystrokes, and mouse usage
- Importing a 3D data cube into a mesh object
- Using the node compositor to build a sequence of input/output interfaces to render our data.
- Using a bit of boilerplate Open Shading Language (OSL) in C with our nodes
- Adjusting our path tracing samples for optimal rendering
- Switching the GUI to real-time rendering with Cycles.
- Setting up a wireframe box around our data
- Determining the camera positioning, movement, field-of-view, and keyframes
- Render output as a still image or animation.

The following steps will set up this visualization.



Your default screen in Blender will show the scene above - a solid cube mesh, camera, and lighting element. One can switch to different topical views with the tabs at the top. In the central viewer, the left mouse button selects objects (surrounded in an orange outline), and the middle mouse wheel can zoom, or rotate with a click. Shift-middle mouse wheel allows translation of the view.

To lock the user view to a camera, left click to select the camera and highlight it in orange. Pop out the Transform controls on the right hand side of the viewer, select View, and check Lock Camera to View.

Data Formatting

The datacube for this example consists of 100 slices in TIF format, but any image or numerical array data format can be used. The data slices should be sequential and numbered with leading zeroes - image001.tif,

image002.tif, image003.tif, etc. Example data from a CT scan can be obtained from http://graphics.stanford.edu/data/voldata/

We will read in the datacube volume as a 3D texture using the Open Shading Language (OSL; https://github.com/AcademySoftwareFoundation/OpenShadingLanguage). Boilerplate examples can be modified to suit our needs.

On the Scene panel, we will change our Render Engine to Cycles and check the Open Shading Language box.

Switch to the Scripting interface, start a new file, name it datacube_osl, and use the following OSL code. This will be used to create a script node in our compositor. The header file is included with Blender, so there should be no need for installation.

```
#include "stdosl.h"

shader volumeTexture(
      point Vector = P,
      string directory="//",
      string prefix="",
      int framePositions=3,
      string extension=".tif",
      float zMax = 2.0,
      int fromFrame = 1,
      int toFrame = 100,
      output color colorData = 1,
      output float nonColorData = 1,
)
{
      // Recenter the position around the object center to -1, 1 to 0, 1
      vector pos = 0.5 * (Vector + vector(1.0,1.0,1.0));

      // Calculate the image number
      int imageNum = (int)floor((1.0 + toFrame - fromFrame) * (pos[2] / (zMax *
0.5)));
      int imageNum2 = (int)ceil((1.0 + toFrame - fromFrame) * (pos[2] / (zMax *
```

```
0.5)));
      float pixelFraction = mod((1.0 + toFrame - fromFrame) * pos[2] , (zMax * 0.5));

      // Make a format string from the given length (ie "%03d")
      string frameFormat = concat( "%0", format( "%d", framePositions ), "d" );
      // Format the current frame/image number (ie "003")
      string imageFrame = format( frameFormat, toFrame - imageNum );
      string imageFrame2 = format( frameFormat, toFrame - imageNum2 );
      // Make the full file name
      string fileName = concat( directory, prefix, imageFrame, extension );
      string fileName2 = concat( directory, prefix, imageFrame2, extension );

      // Get the color from the texture
      colorData   = texture3d( fileName, pos);
      float nonColorData1 = texture3d( fileName, pos);
      float nonColorData2 = texture3d( fileName2, pos);
      nonColorData = nonColorData1 * (1-pixelFraction) + nonColorData2*pixelFraction;
}
```

The OSL script will be executed by one of the nodes in our compositor.

Node Compositor and Shading Editor

The node compositor gives the user a visual methodology of programming their visualization. We set up each node as a series of inputs, parameter settings, and outputs. These are connected together to a final volume rendering.

Select the Compositing GUI menu from the tabs at the top of the screen. Choose the Shader Editor and make sure the “Use Nodes” option is checked. Nodes will be placed into the Compositor via the Add menu as follows:

- **Input → Texture Coordinate.** Currently selected object (our cube) will be the output

- **Script.** We select our datacube_osl script which will translate our OSL function inputs and outputs into GUI elements. Set the directory, prefix, and extension strings to match our data directory. Input will be connected to the Vector variable. zMax has a default of 2, and fromFrame and toFrame should match the number of sequential files in our data directory. We will output our cube information as nonColorData.
- **1st Converter Math node → Multiply.** This will act as an amplifier for the data into our two transfer function controls. The value slide will allow us to fine tune the appearance of our data cube.
- **Converter → ColorRamp** (two nodes). The output from our first Converter will go into two ColorRamp nodes that act as controls for our transfer functions. The first ColorRamp controls the transparent Alpha channel so that we can render the noise into the background and allow the signal emission of our data cube (the object we want to look at) to transfer through.
- **2nd Converter Math node → Multiply.** This further amplifies our second transfer function.
- **Shader → Emission.** The first colorRamp will determine the color map and transparency, and be hooked into the Color input. The second colorRamp will be put into the Emission Strength input.
- **Material Output to Volume.** The shader emission output will finally be connected to the Volume input of the final node.

Our default cube mesh object will serve as our data cube container. Turning on the Viewport Shading of the GUI will show the data cube output in realtime. The transfer function can be adjusted accordingly to suit the user's requirements.

Adding a Wireframe Cube boundary

- Add a second cube with Add → Mesh → Cube. Scale the plane with the S key and press TAB to enter Mesh Edit mode. Subdivide the plane five times or more and press TAB one more time to again return to Object mode.
- Add a material to the cube mesh on the Properties panel and add a Wireframe Modifier. The default thickness should suffice. Choose a Material and Emitter color that contrasts well with the background—blue on black usually works well.
- Set the 'Emission' value to 1.5 or higher.

- Set the World tab background horizon color on the Properties panel to black.

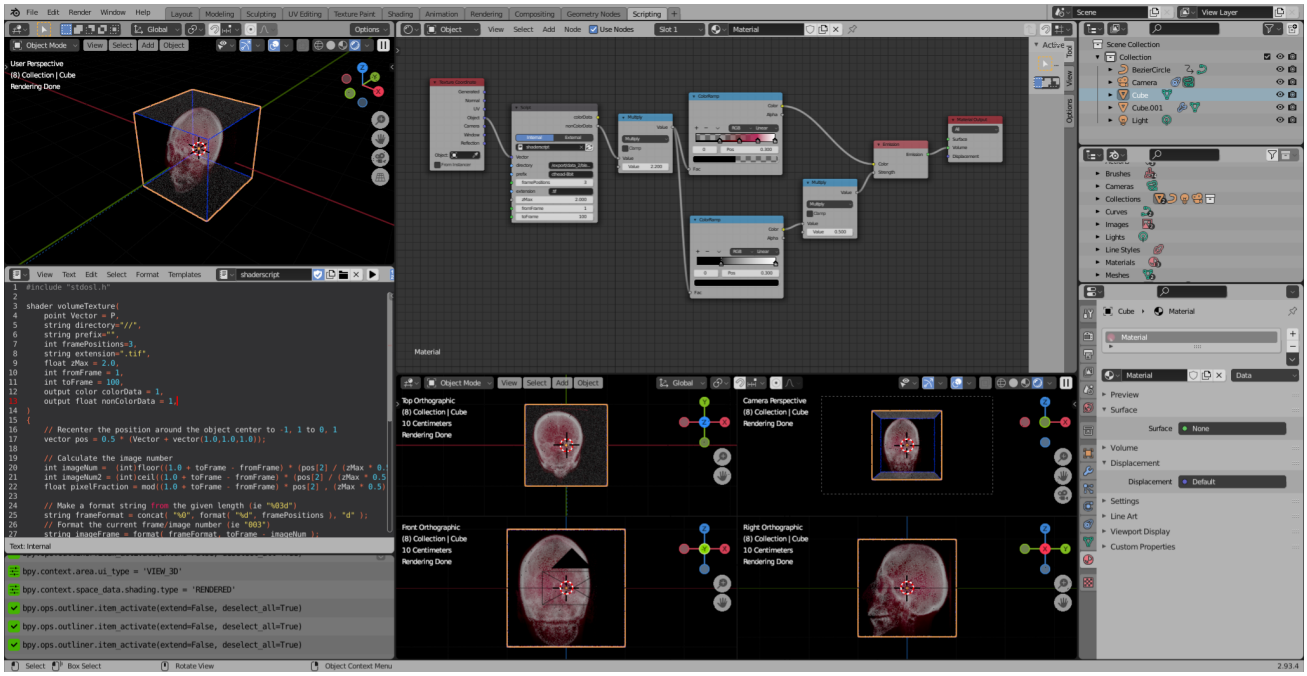
Setting a Camera Path and Movement

We can now use the default camera object to point at an Empty Object. This will allow us to control where the camera points when we set our camera keyframes.

- Add an empty object with Add → Empty → Plain Axes for the camera to track.
- Left-click to choose the Camera object and set the position and rotation values on the Transform toolbar to zero. This will "zero out" the camera at the origin pointing along the -Z axis.
- Add a circular Bezier curve. This will act as a track for our camera to move on, much like a dolly in a movie studio.
- Set three keyframes for the Bezier circle. Start at 0 deg for the Z axis rotation in frame 1, 180 deg at the halfway point, and 360 at the ending frame. In this example, only the rotation needs to be keyframed.
- Switch back to the Camera object. Set an Object Constraint of Follow Path, and choose the Target as the Bezier Circle.
- Set a second Constraint of Track To, and choose the Cube object as the target.
- The camera will orient toward the Cube and follow the circular track during the animation. Hit the Play button at the bottom of the screen on the animation toolbar to confirm this element.

The visualization can now be exported in a 1080p HD video.

- On the Output Properties tab the Properties panel and make sure the resolution is 1920x1080p. Set the frame rate to 24 or 30 frames per second.
- Set the output to AVI JPEG, quality 100 percent, and specify a unique filename (*.mp4) on your local disk. Click the Render → Render Animation menu at the top of the tab to render the visualization.


File Edit Render Window Help
Layout Modeling Sculpting UV Editing Texture Paint Shading Animation Rendering Compositing Geometry Nodes Scripting
User Perspective
(8) Collection | Cube
Rendering Done
Material
#include "stdosl.h"
shader volumeTexture(
point Vector = P,
string directory="//",
string prefix="",
int framePositions=3,
string extension=".tif",
float zMax = 2.0,
int fromFrame = 1,
int toFrame = 100,
output color colorData = 1,
output float nonColorData = 1,
)
{
// Recenter the position around the object center to -1, 1 to 0, 1
vector pos = 0.5 * (Vector + vector(1.0,1.0,1.0));
// Calculate the image number
int imageNum = (int)floor((1.0 + toFrame - fromFrame) * (pos[2] / (zMax * 0.
int imageNum2 = (int)ceil((1.0 + toFrame - fromFrame) * (pos[2] / (zMax * 0.5
float pixelFraction = mod((1.0 + toFrame - fromFrame) * pos[2] , (zMax * 0.5)
// Make a format string from the given length (ie "%03d")
string frameFormat = concat("%0", format("%d", framePositions), "d");
// Format the current frame/image number ie "003"
string imageFrame = format(frameFormat, toFrame - imageNum);
Text: Internal
bpy.context.area.ui_type = 'VIEW_3D'
bpy.context.space_data.shading.type = 'RENDERED'
bpy.ops.outliner.item_activate(extend=False, deselect_all=True)
bpy.ops.outliner.item_activate(extend=False, deselect_all=True)
bpy.ops.outliner.item_activate(extend=False, deselect_all=True)
Top Orthographic
(8) Collection | Cube
10 Centimeters
Rendering Done
Camera Perspective
(8) Collection | Cube
Rendering Done
Front Orthographic
(8) Collection | Cube
10 Centimeters
Rendering Done
Right Orthographic
(8) Collection | Cube
10 Centimeters
Rendering Done
Scene Collection
Collection
BezierCircle
Camera
Cube
Cube.001
Light
Brushes
Cameras
Collections
Curves
Images
Lights
Line Styles
Materials
Meshes
Preview
Surface
Surface None
Volume
Displacement
Displacement Default
Settings
Line Art
Viewport Display
Custom Properties
Select
Box Select
Rotate View
Object Context Menu
2.93.4

Blender resources:

Open Shading Language (OSL)

Blender OSL documentation

Tutorial: https://thhube.github.io/tutorials/osl/osl.html

BlenderGuru

Blender Materials

BlenderDiplom

BlenderNation

CG Cookie

Tutorials and Modelling

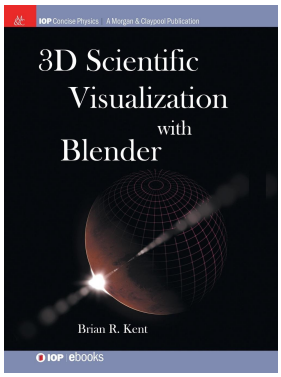
Wiki Book

Blendtuts

Blender Models

Videos and Tutorials:

https://www.youtube.com/user/VisualizeAstronomy/videos



Publications:

3D Scientific Visualization with Blender:

https://iopscience.iop.org/book/978-1-6270-5612-0

Visualization in Astronomy:

https://iopscience.iop.org/journal/1538-3873/page/Techniques-and-Methods-for-Astrophysical-Data-Visualization

Extragalactic Distance Database:

https://ui.adsabs.harvard.edu/abs/2009AJ....138..323T/abstract

360 Video and Spherical Panoramas:

https://ui.adsabs.harvard.edu/abs/2017PASP..129e8004K/abstract