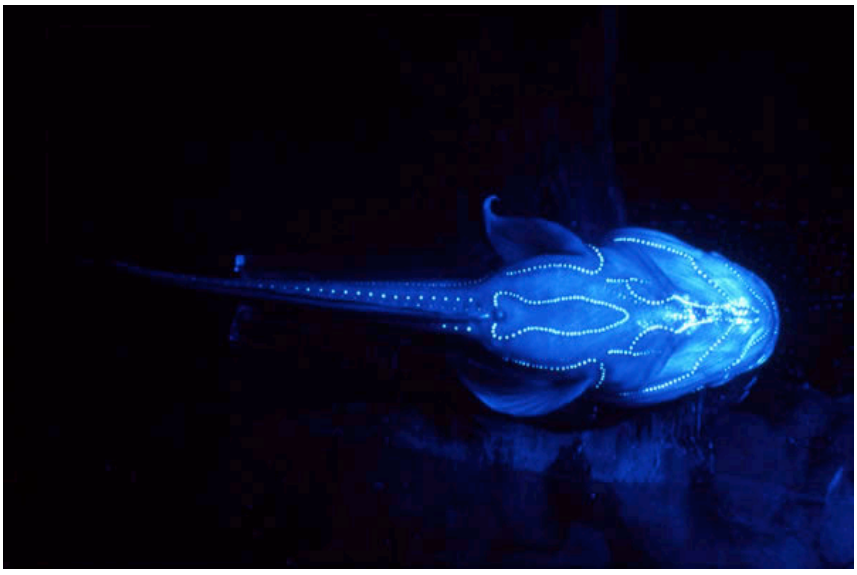
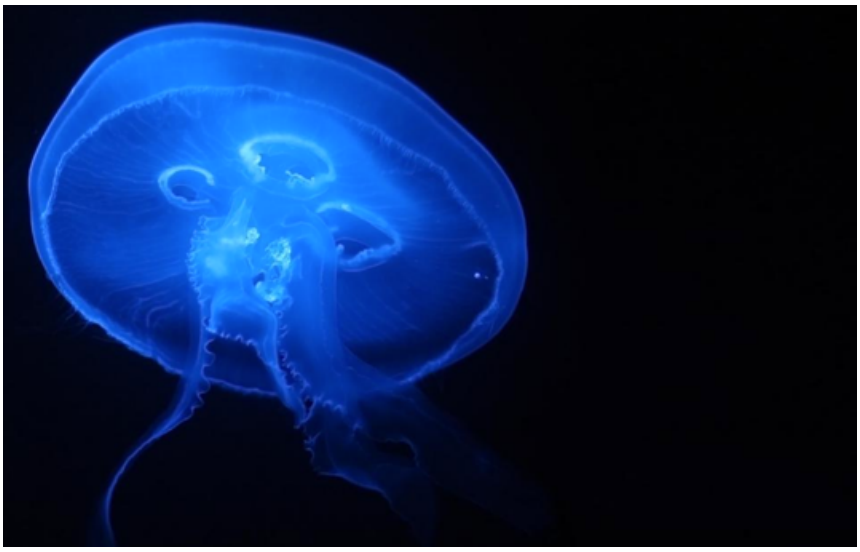
# Bioluminescence

Chris Fontas & Forrest Browning

## Introduction

Our goal for the final project was to render a bioluminescent organism. Bioluminescence is a type of luminescence (cold light) resulting from biochemical reactions in an organism. Many forms of life have this property, but we decided to render a deep-sea fish, many of which are well known for their beautiful blue glow.





# Implementation

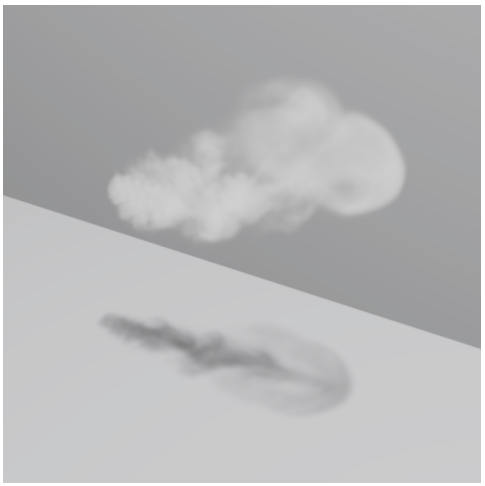
**Part 1a: Multi-scattered Light with Volumetric Photon Mapping**

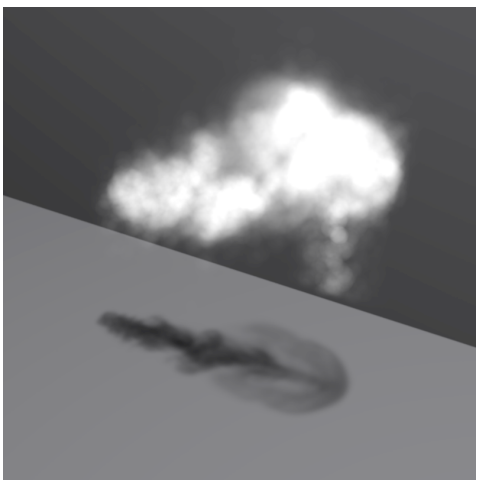
After a lot of initial research, we concluded that the best way to model the bioluminescent glow we were after was by means of a participating medium. PBRT comes equipped only with single-scattered volumes, which is fine for things such as smoke which are rather dull in color, but our volume needed a fullness to it that could only be achieved with a multi-scattering algorithm.

We decided to use the volumetric photon mapping technique described in Henrik Wann Jensen's book, *Realistic Image Synthesis Using Photon Mapping.* Volumetric photon mapping works similarly to photon mapping used for indirect illumination and caustics, however there are a few key differences. First, when a photon interacts with a medium it is either absorbed or scattered according to a phase function (not a BRDF) where the integral of the probabilities for scattering over all possible directions (the unit sphere) must equal one. Furthermore, with caustic/indirect photons, the radiance of each photon is determined by scaling its intensity by the BRDF of each intersected surface, but for volumes the radiance is determined by the coefficients for absorptivity, scattering and termination of the medium (sigma_a, sigma_s and sigma_t respectively) and not the phase function. Finally, as each eye-ray marches through the medium, the radiance contribution of the k-nearest photons to each ray segment is added to the radiance due to direct illumination.

To implement this algorithm in PBRT, we first had to extend the photon map class to shoot volumetric photons in a separate loop from other photon types as well as store them in their own KDTree. Then in the single scattering integrator, we modified the radiance calculating function to include a search of our volume KDTree and added the photon radiance to return value.

Our results are below. The image on the left is a cloud rendered with single scattering, while the one on the right was rendered with our multi-scattering algorithm. The multi-scattered cloud has bright specular highlights and is much more "wispy" than the single-scattered cloud.

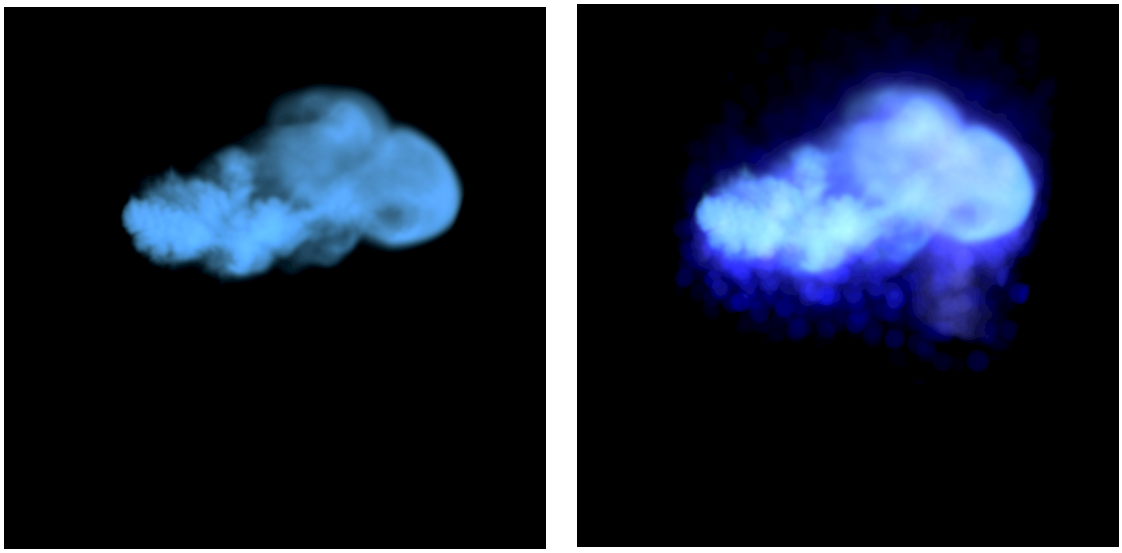




**Part 1b: Adapting the material properties**

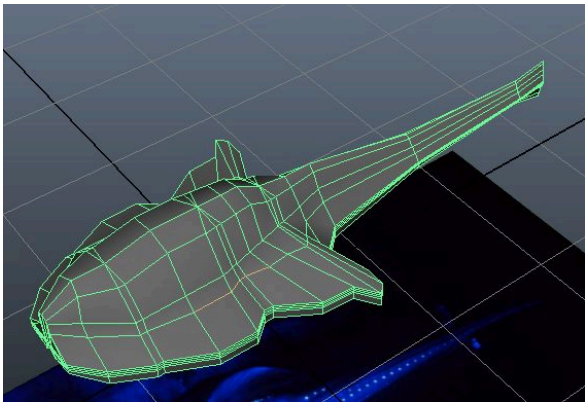
After we finished implementing the multi-scattering algorithm, it was time to modify the medium's physical properties to change it from a simple white cloud to a blue one with a neon glow. To achieve this effect we had to both change the color of the cloud itself to be blue as well as modify the extinction coefficient sigma_t so that the majority of absorbed light would be in the red and green color channels.

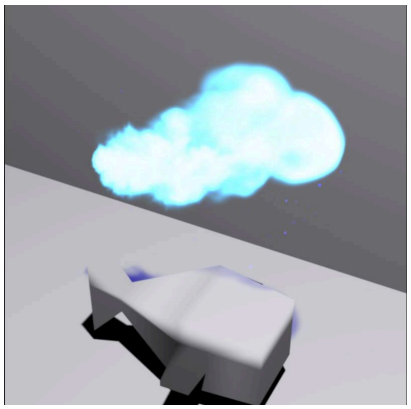
Our results are below. The image on the left shows a blue cloud with single-scattering only while the one on the right is the cloud rendered with our multi-scattering algorithm. The multi-scattered cloud has not only a nice translucent glow to it but also a nice blue "halo" of photons around the sides which definitely adds to the effect.



**Part 2: Modeling the organism**

The mesh was fully built in Maya. Using Maya's project image plane option, we were able to take the reference photo of the bioluminescent fish and place it in the Maya scene a large, flat plane. This is a fairly standard procedure used by artists to allow for a more accurate and true-to-life model. We then started with one large rectangular prism that was roughly the size of the fish, and added more detail until we had the final product you see in our image. Below are images of our mesh in mid-production, with the reference image used visible in the picture on the left.





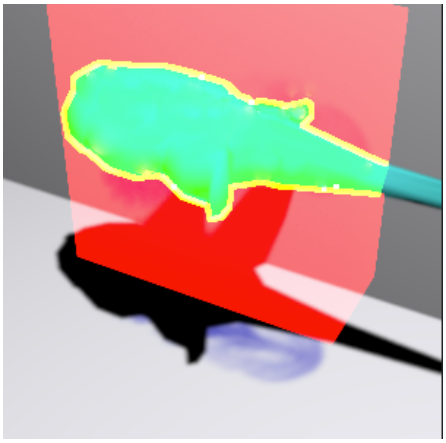
**Part 3: Confining the shape of the medium**

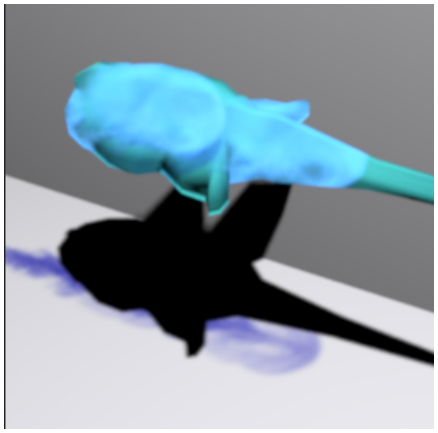
The last step in our process was to contain the medium within the bounds of our mesh. PBRT only confines the extent of a volume using an axis-aligned bounding box which would not suit the irregular geometry of our mesh.

To get around this, we placed the mesh entirely within the frame of the medium's bounding box. Then in the volume integrator, for every ray segment marching through the medium, we test to see if the segment point lies within the volume encapsulated by our mesh. If not, the radiance contribution from that location is ignored. Otherwise, it is added to the final contribution.

The test we used was very simple. For every point within the volume, we shoot a ray in a random direction starting at that point and count how many times it intersects our mesh. An even number of intersections (including 0) mean that the point lies outside the mesh, while an odd number means that it is inside.

The images below show various stages along the way of confining the medium. In the image on the left, red points are those outside the mesh, green are inside and yellow is simply due to the color mixing of red and green along the edges. The image on the right shows the medium overlaid with a diffuse version of our mesh.





# Problems Encountered

There were several challenges to getting our implementation just right. First, even after we got our volumetric photon mapping algorithm working, tweaking the settings to get the cloud to look just right took a lot of time and effort. There were a lot of variables at play (cloud color, sigma coefficients, number of photons to shoot, bounce limit, photon lookup search radius, etc) which each affected how the cloud came out.

Afterwards, we also ran into a lot of difficulty importing our Maya mesh into our PBRT scene file. We fully built the mesh in Maya, and then exported it to an .obj. Once in Obj form, we imported it into Blender and then immediately exported it right back to obj. The Blender step was necessary for us, because Maya doesn't to have as comprehensive of obj export options. When exporting from Blender, we had it triangulate the mesh (which wasn't already done by Maya), and set various options regarding UVs. Once in an acceptable obj form, we then had to convert it to PBRT triangle meshes, for which we used a python script written by students in the class in 2011 (https://graphics.stanford.edu/wikis/cs348b-11/olliver/Final_Project?action=AttachFile&do=get&target=obj2pbrt.py).

Surprisingly, the hardest part of the project was confining the volume to the shape of the mesh, which we assumed would be the easiest part.

# Division of Labor

**Chris**:
Chris happened to already own the Henrik Wann Jensen book and so he was responsible for reading through it and getting a firm understanding of the theory before implementation began. Chris also did online research to try and find other articles that may have been of use.

**Forrest:**
As Forrest already had a good deal of experience modeling 3D meshes using tools such as Maya, creating the final fish mesh (seen in our final image) was left primarily to him. Additionally, he handled the conversion of the Maya mesh into a PBRT compatible format (not that easy!).

**Both:**
The majority of the coding was done together. Due to the complicated math present in the algorithms used, it was useful to reason out the solution together and fill in the gaps in each other's understanding which helped us get a correct implementation working relatively quickly. Furthermore, towards the end, most of our time was spent tweaking render settings to find the optimal combination of values so it was important to have both of us together to judge.

# Final Result

Final image took around 25-30 minutes to render at low resolution (400x400). We also turned on PBRT's caustic photon map in order to get some caustics on the floor underneath the fish.