Relativistic Ray-Tracing: The Appearance of Rapidly Moving Objects

Team Einstein Members:

- Jeremy Daniel
- Cyrus A Dolph V
- Jean-Emile Elien

Abstract:

Team Einstein's project was an attempt to modify an existing raytracer to model the effects of high-speed motions on objects, including spacetime intersections, relativistic transformations and Doppler shifts. Partial success was achieved: the modified raytracer correctly renders these three effects in boxes and spheres. Other object primitives remain to be modified, and the raytracer doesn't properly handle shading, reflection, or refraction.

Introduction:

In "classical" ray-tracing, the paths of light followed are the ones that start at the eye, and land on an object. We collect the possible contributions of color from a point by tracing more rays in likely directions. This recursive process continues until it "bottoms-out", either by too little contribution, or some maximum recursion depth.

This process is independent of the velocity of the objects. If we took a scene with moving objects, and opened the camera for an instant, and recorded the image produced, the ray-tracing algorithm would have produced the same image. There is no dependence on the relative speed of the objects. Now, it is also possible to have the same image if we assume that the speed of light is infinite. It no longer matters how fast the objects are moving, the instant the camera eye is opened, all of the rays can be traced out from the start points and accumulate the image. So, an interpretation of "correctness" for the classical ray-tracer is the assumption that the speed of light is infinite. Our project is an attempt to remove that assumption: to see what, if any distortions occur if the speeds of the objects are comparable to the speed of light.

The effects should be dramatic because when the objects are traveling that fast, the time it takes for the light to reach the eye is also enough time for the object to move. Thus, the light from all points on a sphere, for example, will not reach the eye at the same time, thereby causing a warping of the viewable object.

Another effect that should be noticed is the Doppler shift, the variation in wavelength of light as a function of the speed of an object. To an observer moving forward at high speed, the apparent color of a stationary object will be red-shifted, as the wavelength of light from that object appears contracted. Likewise, if an observer is moving away from the object at high speeds, the objects appears blue-shifted.

Goals:

We chose POV-Ray as the ray-tracer that we would use to add our effects. POV-Ray is a ray-tracer to which the source is freely available, and compiles on most Unix platforms. Since our members would be working on both Linux and SunOS, POV-Ray seemed better than ivray.

Our team had several goals in the modeling the finite speed of light:

- Intersections in time: the methods of intersection currently assume static objects. A ray fired by our ray-tracer will be traveling back in time to intersect an object. It is not known if that object is even there, much less where it hits. So, a new method to compute intersections in time is needed.
- Lorentz transformation: The measurement of space and time changes when we take into account the fact that the speed of light is the same for all inertial observers. This means the directions of the rays will change just from the frame of reference of the camera to the frame of reference of the moving object. So transformations are needed in the renderer to account for this fact.
- Doppler shift of light: when determining the apparent color of a point in the scene, it becomes necessary to account for the relative velocities of the objects involved to compute the Doppler shift. Since, POV-Ray uses an RGB color model, we must develop a method for representing the actual light spectra associated with colors. Then we may apply the Doppler shift equation to the wavelengths involved, and convert the results into a new RGB color.

Achievements:

Intersections with moving objects

In considering the intersections with moving objects, we narrowed our focus to motions of constant velocity. Given this restriction, the resulting formula for intersections in spacetime turned out to be rather simple. The case of dynamic intersections can be reduced to that of static intersections by just adding the velocity of the object to that of the ray. Given this new ray, it is possible to compute the point of intersection if the target object is moving. To show this, consider intersections with a moving sphere:

The velocity of the sphere is some constant *v*, and its position at *t=0* is *P*. There is also a ray, which originates at point *R*, and moves with a velocity *c* which is finite. So, the equations of motion describing these objects are:

$$\vec{P}(t) = P_0 - \vec{v}\,t$$
$$\vec{R}(t) = R_0 + \vec{c}\,t$$

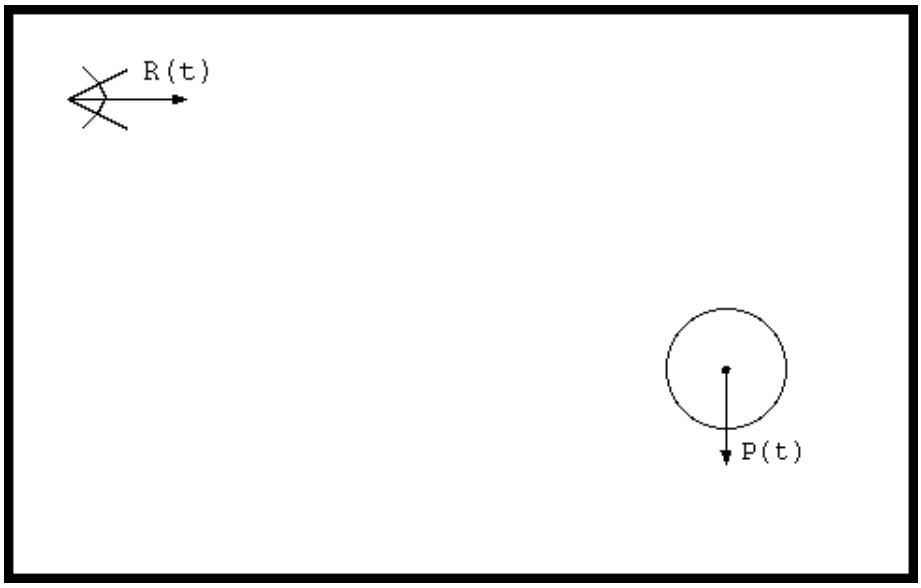


Figure 1 - Intersect Ray with Moving Sphere

Notice the minus sign in the equation for *P(t)*. It is because as we trace forward in *t*, we are really tracing backwards in time, and the motion of the object is reversed. First, consider the situation of a static sphere: *v=0*. Now, taking the equation of a sphere, we solve for *t*:

$$\begin{aligned}
\vec{P} \cdot \vec{P} &= r^2 \\
(\vec{P} - (\vec{R} + \vec{c}\,t)) \cdot (\vec{P} - (\vec{R} + \vec{c}\,t)) &= r^2 \\
((\vec{P} - \vec{R}) - \vec{c}\,t) \cdot ((\vec{P} - \vec{R}) - \vec{c}\,t) &= r^2 \\
\|\vec{c}\,\|^2 t^2 - 2(\vec{P} - \vec{R}) \cdot \vec{c}\,t + \|\vec{P} - \vec{R}\|^2 - r^2 &= 0
\end{aligned}$$

Now, if we allow a finite velocity for the sphere, we get:

$$\begin{aligned}
((\vec{P} - \vec{v}\,t) - (\vec{R} + \vec{c}\,t)) \cdot ((\vec{P} - \vec{v}\,t) - (\vec{R} + \vec{c}\,t)) &= r^2 \\
((\vec{P} - \vec{R}) - (\vec{v} + \vec{c}\,)t) \cdot ((\vec{P} - \vec{R}) - (\vec{v} + \vec{c}\,)t) &= r^2 \\
\|\vec{v} + \vec{c}\,\|^2 t^2 - 2(\vec{P} - \vec{R}) \cdot (\vec{v} + \vec{c})t + \|\vec{P} - \vec{R}\|^2 - r^2 &= 0
\end{aligned}$$

Notice that they are quite similar. In fact, the only difference between the two equations is the addition of the velocity of the sphere. So, if we considered a new "ray" with the direction and magnitude *c + v*, then we can perform the original ray-sphere intersection to determine whether the sphere is hit! The intuition is to add the velocity of the object to the outgoing ray, and compute the standard intersection. If they hit, the moving object will also hit the ray. This significantly simplifies the process of trying to do intersections, because all of the algorithms for doing fast intersections still work, it just requires a tweaking of the intersecting ray to be correct. There is only one problem: the point of intersection is not the correct point in spacetime where we *actually* hit the object. We have the relative position on the sphere, but not the point in camera space.

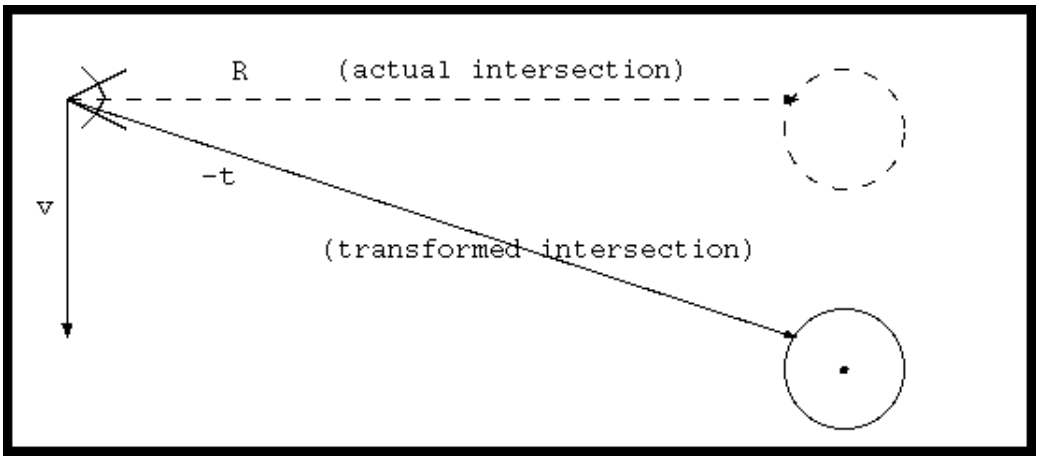


Figure 2 - Transform Ray to Sphere Frame

Fortunately, it is still possible to find the correct point of intersection. Now, we have the value t_I, that corresponds to the intersection of the modified ray and the sphere. However, t_I is also the time it would have taken the *original* ray to hit the *moving* sphere! To see this, we resolve the vector into its component parts:

$$\vec{R'}(t) = \vec{R_0} + (\vec{v} + \vec{c}\,)t = \vec{R_0} + \vec{c}\,t + \vec{v}\,t = \vec{R}(t) + \vec{v}\,t$$

If we resolve the composite ray into its components: *v* and *c*, a time-step along the new vector is the same as a time step along each component vector. The resultant vector that connects the intersection point and the origin of the ray is the composite ray scaled by t_I. That is equivalent to scaling *v* and *c* by *t*, and adding them together. This can also be seen as having the sphere travel with time $-t_I$ (forward in time) with velocity *v*, which is exactly what we are trying to solve. So, in order to contruct the "correct" intersection point, take the original ray, and plug in the value of *t* for the computed intersection.

$$I = \vec{R}(t_I) = \vec{R} + \vec{v}\,t_I$$

So, it is possible to compute the intersections with an object moving with constant velocity, by adding its velocity to the ray, computing the intersection, and using the resultant time to add to the velocity of original ray to compute the intersected point.

Observable effects

Just taking this intersection into account leads to some a very interesting visual effect, the apparent "stretching" of an object. To see why, look at the following diagram:

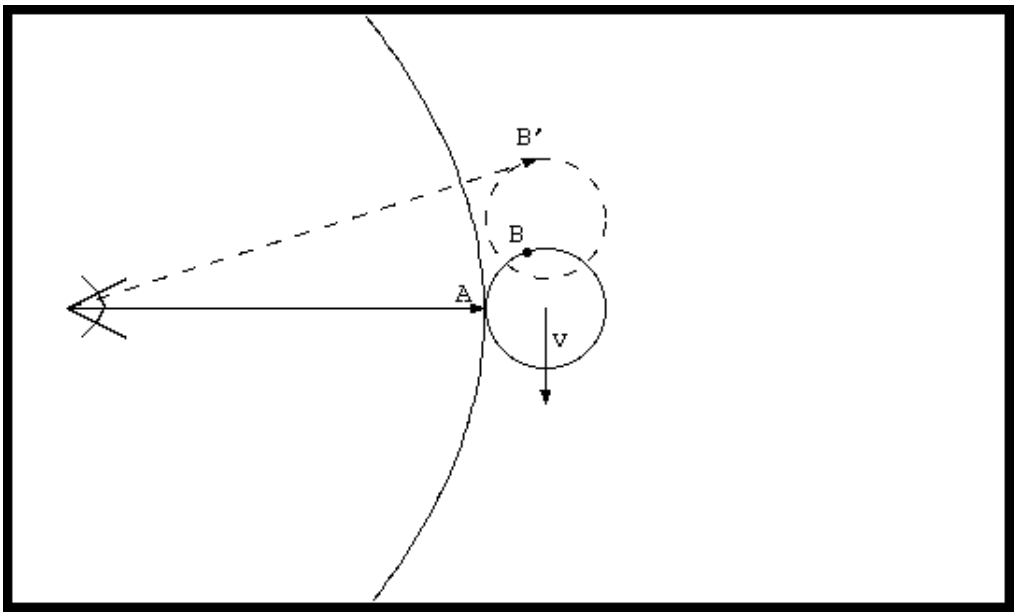
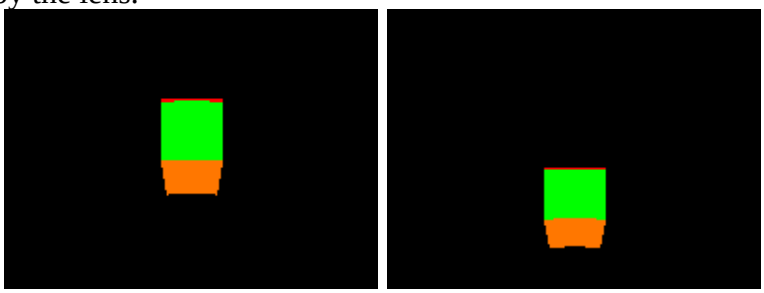


Figure 3: Observed points and light-sphere

We have a ball moving at some high velocity, and a stationary camera. The question is: At some instant the camera opens its lens, what does the camera see? Let's say that point *A* reaches the camera the instant that it opened the lens. That defines a sphere with a radius that is the distance separating *A* from the camera. Now, since the speed of light is finite, only photons currently on the "light-sphere" can reach the camera. So, when does point *B* seem to appear? It had to start further back in time (*B'*) to reach the light-sphere in time to start at the same time that *A* starts. But, since that is further back than *B*, the sphere seems stretched. At very high speeds, the object appears to rotate. To see this, look at point *B* in the diagram. Normally this point wouldn't even be seen by the object, but when a photon is emitted from that point, the rapidly moving object actually "gets out of the way" of the photon heading towards the camera and is caught by the lens.



Two boxes rendered to be traveling up at .7c in the *y* direction. When the boxes are stationary, only the green face is seen, but when they are moving, the bottoms of the objects are visible as well. Note that the right image, the relativistically corrected one, has a shorter face. That is the contraction in length, that makes it look more rotated than the left image, which looks sheared.

Lorentz Transformations

The Lorentz equations are derived from the assumption in Special Relativity that in all inertial frames, the speed of light is the same. The canonical formulation has two frames, *S* and *S'*, oriented so that the respective origins intersect at $t=t'=0$. The frame *S* is also moving in the $+x$ direction with speed v. The

equations that govern the measurement of an *event* in *S'* given an event in *S* is given by: (in all equations, if *c* is missing, it is assumed 1)

$$\begin{aligned} \beta &= \frac{v}{c} \\ \gamma &= \frac{1}{\sqrt{1-\beta^2}} \\ ct' &= \gamma(ct - \beta x) \\ x' &= \gamma(x - \beta ct) \end{aligned}$$

In the general case, the frame *S'* is moving with some velocity *v* in any direction relative to *S*. The resulting transformation for the events in *S* to *S* is:

$$\begin{aligned} ct' &= \gamma(\, ct + \vec{\beta} \cdot \vec{x}\,) \\ \vec{x'} &= \vec{x} + (\gamma - 1)\frac{(\vec{\beta} \cdot \vec{x})\vec{\beta}}{\vec{\beta}^2} + \gamma\vec{\beta}ct \end{aligned}$$

We also need the equation that transforms rays from one frame to another. To derive the expression, take the general Lorentz equations and divide *x'* by *t'*. The resulting equation, after some algebra, looks like:

$$\vec{u'} = \frac{\vec{u} + (\gamma + (\gamma - 1)\frac{\vec{\beta} \cdot \vec{u}}{\vec{\beta}^2})\vec{\beta}}{\gamma(1 + \vec{\beta} \cdot \vec{u})}$$

This equation takes a ray from *S'* to *S*. To get the inverse, simple reverse the sign of the velocity. This ray corrects the incoming ray from the camera in order to intersect the object properly.

Observed effects

As far as object transformation is concerned, this actually is much less interesting than simply considering the case of the finite speed of light. Taking Figure 3 above, if Special Relativity were taken into account, the sphere would be Lorentz contracted. The sphere, however appears elongated from the explanation above. So, the net result leaves the sphere looking exactly as it did before, but with a rotation. On the example of the single sphere moving past the observer, the spheres actually look like spheres, but rotated. This correlates to the Weisskopf paper written in 1960 that describes this experiment.

This finding is very difficult to correlate with the other papers that have been released in the past 10 years. None of them actually use the example of the 1960 paper, so it is difficult to make a direct comparison.

Limitations

We ran into several visual problems when rendering the scenes:

1. Shading never seemed to work correctly. When we had an object moving at some high speed, the points corresponding to the "back" of the objects never seemed to be shaded correctly. We believe this is a problem of the shading algorithm using the normals in object space with the ray in camera space instead of the ray in object space. Because of this problem, we had to use ambient objects and no light sources to view certain scenes properly.
2. Taken separately, the camera can move, and the objects can move, but not both simultaneously. The camera is not an object in POV-Ray, and the first ray spawned per pixel is special. So, Doppler shifting that should be seen if the camera was moving will not happen. This could be handled, there was just little time.
3. There is also the effect that the intensity of light is a function of the relative velocity of the object, so it not only red-shifts when moving toward the observer at high speed, it should brighten. We did not model that in POV-Ray, but we think that might be handled in the Doppler shift code, as red light is inherently brighter than blue light. However, we are not sure that the inherent intensity alone will account for the change.

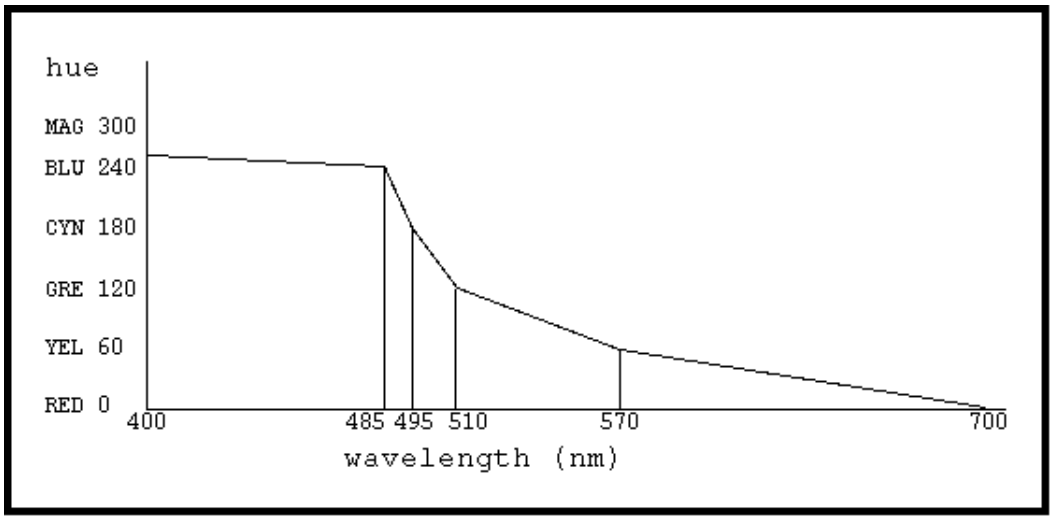
Doppler Shift

To compute the Doppler shift, we apply the following relativistic Doppler shift equation:

$$\lambda = \lambda_0 \frac{1 + \vec{v} \cdot \hat{r}}{\sqrt{1 - \|\vec{v}\|^2}} = \lambda_0 \gamma (1 + \vec{v} \cdot \hat{r})$$

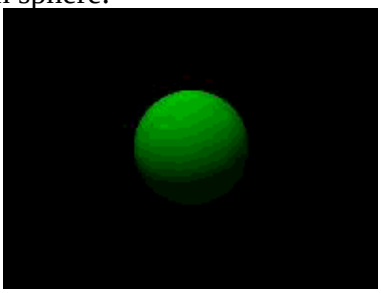
where *r* is the ray direction vector, and *v* is the velocity of the intersection point relative to the source of the ray.

The ray direction and relative velocity are already available, as they have been found by the intersection test. The challenging problem of implementing the Doppler shift is finding the wavelength values to substitute into the equation. We require a practical method of determining the wavelengths of light associated with an RGB color. This is complicated because there are many different spectra that correspond to each single color - how do we choose one given an RGB triple? It is not possible to represent colors such as magenta as single wavelengths, as magenta is always a combination of red and blue light. The approach we took was to treat each color as representing light consisting solely of its associated red, green, and blue values. Each RGB component is converted into a HSV representation using a standard algorithm, and the wavelength is then extracted from the hue component using the conversion function described in the following graph:



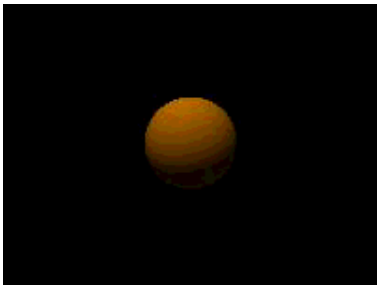
We apply the Doppler shift equation to each of the three wavelengths, and then convert them back into new RGB values using the reverse of the process described above. The final result is the summation of the three Doppler shifted wavelengths.

Here is an image of a stationary green sphere:



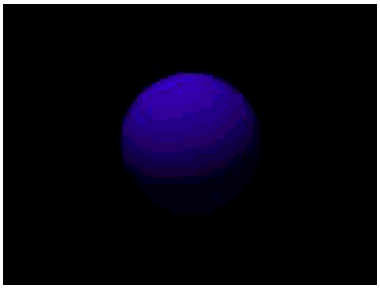
If the sphere is moving towards the camera, it will shift towards red:

v = -0.1c: 

v = -0.2c: 

If the sphere is moving away from the camera, it will shift towards blue:

v = +0.1c: 

v = +0.2c: 

As can be seen in these images, the amount of doppler shifting is dramatic, even at low velocities. When the velocity exceeds 0.3 c, the entire image will be shifted out of the visible spectrum! For this reason, we have implemented a scale factor variable to the scene description language, so that the amount of doppler shifting can be adjusted to obtain interesting pictures of scenes involving objects travelling at very high velocities.

Individual Contributions:

Cyrus' primary responsibility was the implementation of the Doppler shift in POV-Ray. This involved researching colorimetry and writing code to convert RGB colors into wavelengths as well as computing the correct shift amount. As Cyrus had prior experience with POV-Ray, he helped the group learn to use POV-Ray's scene description language at the beginning of the project, and contributed to the effort of understanding the source code.

Jean-Emile was responsible for the deriving intersections of the objects and rays in spacetime, as well as for applying the Lorentz transformations to the rays when they intersected the objects. Involved was the formulation of the inversion of a 5x5 matrix. It turned out that it was useful to "turn off" Special Relativity to see the stretch that objects undergo. So, in order to simplify the task of adding the effects of Special Relativity to the scenes, Jean-Emile made relativity an option that could be specified in the scene descriptions.

Jeremy's chief responsibility was in finding a way to make animations out of the POV-Ray scenes. In addition some work was done to get POV-Ray to compile on athena. Jeremy was also responsible for the version control system for the team, and general locker management issues.

All three team members collaborated in writing this paper.

Lessons Learned:

The biggest challenge of our project was understanding the five megabytes of POV-Ray source code well enough to perform extensive modifications. We discovered that POV-Ray had many optimization features such as automatic bounding boxes and vista buffers which needed to be circumvented, as they no longer worked with relativity. We were attracted to POV-Ray because of its large feature-set, fast rendering speed, multi-platform support, and its selection by other researchers for modification for similar projects. We would probably have been wiser to choose a smaller raytracer to modify, such as Ivray from problem set 4, since we were already very familiar with its source code. This would have allowed us more time to enhance our project with additional features, such as reflection and shadows.

Our group worked reasonably well together, although we didn't meet often enough and there should have been more communication in between meetings, especially over email. We were able to recover almost seamlessly when our fourth team member dropped out at the last minute, which indicates we were well organized.

Bibliography:

- V. F. Weisskopf. "The visual appearance of rapidly moving objects." *Physics Today*, Number 13,

1960.
- Rindler, Wolfgang. *Introduction to Special Relativity*, Oxford Science Publications, Clarendon Press, 1991.
- Foley, James D. et all *Computer Graphics: Principals and Practice* Addison Wesley Publishing. Reading, MA, 1996.
- John Walker's C-ship (*http://www.fourmilab.ch/cship/cship.html*)
- Andrew Howard's Relativistic Ray-Tracing page (*http://www.cs.mu.oz.au/~andrbh/raytrace/raytrace.html*)

Appendix A: Changes to the POV-Ray description language

We added several variables to `global_settings`. I will give an example of their use and then an explanation.

```
global_settings {
  relativity 0
  doppler_shift 1
  doppler_factor 0.07
}
```

- **`relativity`**: turns on or off relativity calculations. Useful for emphasizing the stretching effect caused by the the motion. Has a default value of 1.
- **`doppler_shift`**: turns on or off doppler calculations. Useful if you want to see the original colors of the scene. Has a default value of 1.
- **`doppler_factor`**: scaling factor used for doppler color changes. Since all the light shifts out of the visible when going faster than 0.3 c, this allows you to see the shifting effect, by rescaling the visible spectrum. Has a default value of 1.

In addition we added a velocity component to the descriptors for objects.

```
sphere {
  <1, 1, clock*0.5>, 1
  velocity <0.0, 0.0, 0.5>
}
```

- **`velocity`**: the velocity component of an object is an x y z vector in terms of c. The above sphere would be traveling at half the speed of light in the positive z direction. Has a default value of `<0, 0, 0>`.
- **position**: in order to produce animations in which the objects are moving you need to make the position of the object the integral of the velocity, with whatever initial values you choose. For the above velocity the position would be $<0.0*clock+x_0, 0.0*clock+y_0, 0.0*clock+z_0>$. (Of course factors of zero can be left out.)

As an alternative to the objects moving, all the objects can be made stationary and the camera can be moving. Camera motion is specified the same way as object motion within the `camera` modifiers. Scenes in which both the objects and the camera are specified as moving will produce incorrect images.