Molten Glass

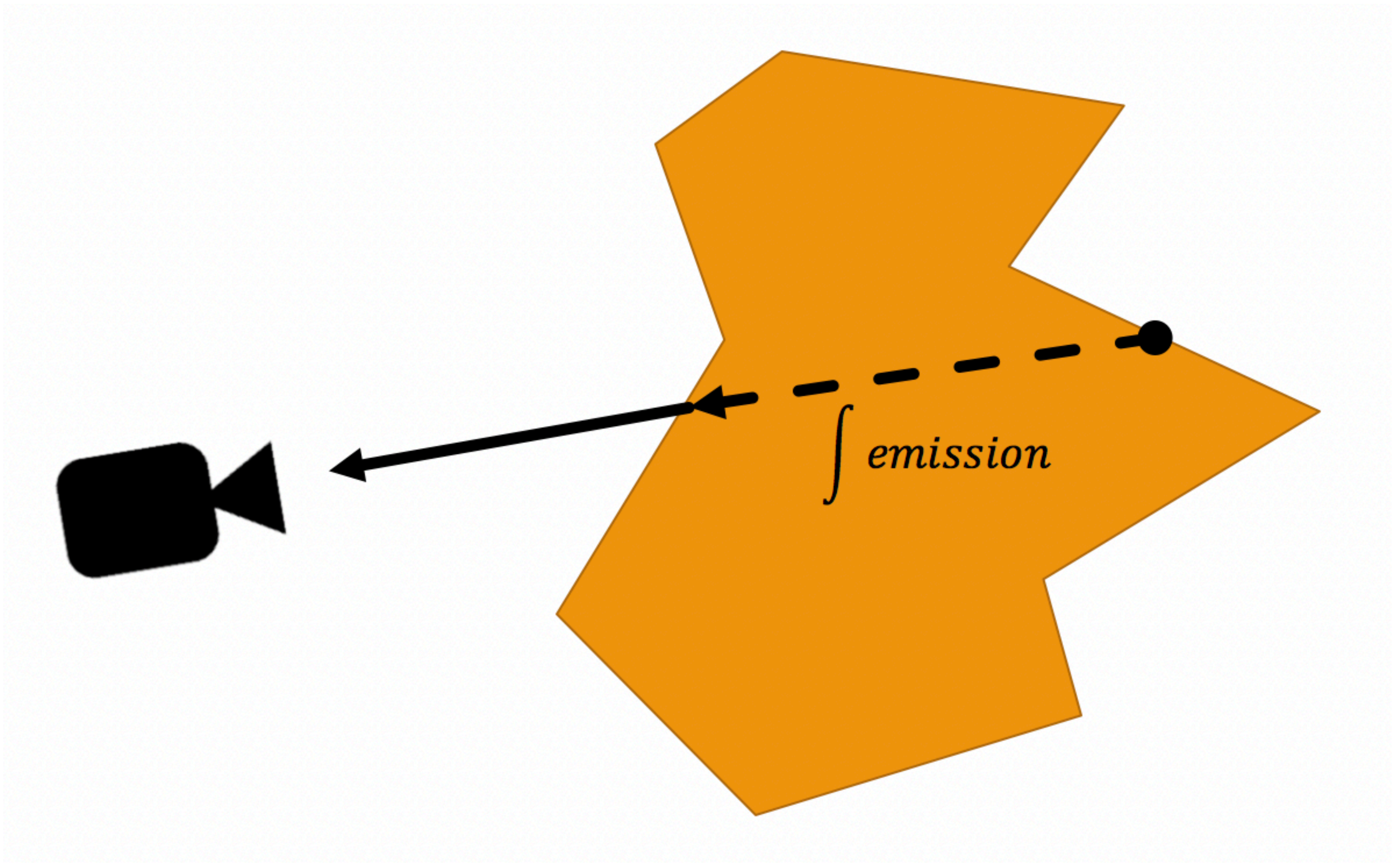
Kyle Fisher

For my project, I set out to create an image in which an artist is creating a beautiful glass sculpture from molten glass. There are many interesting visual properties which can be observed of molten glass, such as its spatially-varying volumetric emission, refraction, and reflection. To achevie these effects simultaneously, I created a new type of light, the "pseudo-volumetric surface light", which allows the user to render a shape with both surface effects and an emissive volume with a single additional line in the scene file **pbrt** declaration. This light is easily integrated into **pbrt**'s existing light sampler, and behaves exactly like an ordinary surface/area light except for how the emission at each point on the surface is calculated.

My Approach

Out of the box, **pbrt** supports volumetric scattering and absorption from media, but not volumetric emission. Because of this, one strategy I considered was extending the **Medium** class to support emission -- then, I could use a glass surface to enclose the medium to get the surface effects needed. However, by enclosing the medium in a refractive surface, it would be extremely difficult to perform accurate light sampling on the emission field. Instead, I came to an alternative solution in which mediums were not needed at all.

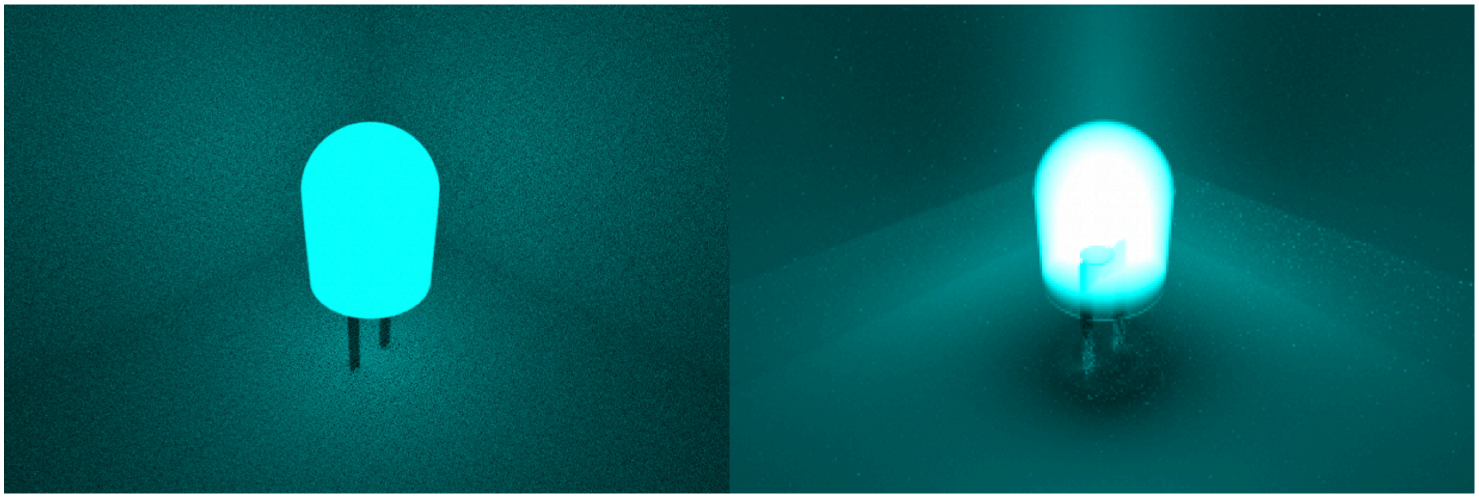
Because molten glass objects are indeed a light source, I decided to extend **pbrt** with a special type of light, **VolumeLight**, which can be used to render arbitrary emissive volumes behind a surface. **VolumeLight** is identical to **AreaLight** (which implements an emissive surface), except that when it is sampled it performs an integration over the interior of the shape. This is nicely summarized in the diagram below:



Sampling, Surface Effects, and Power Estimation

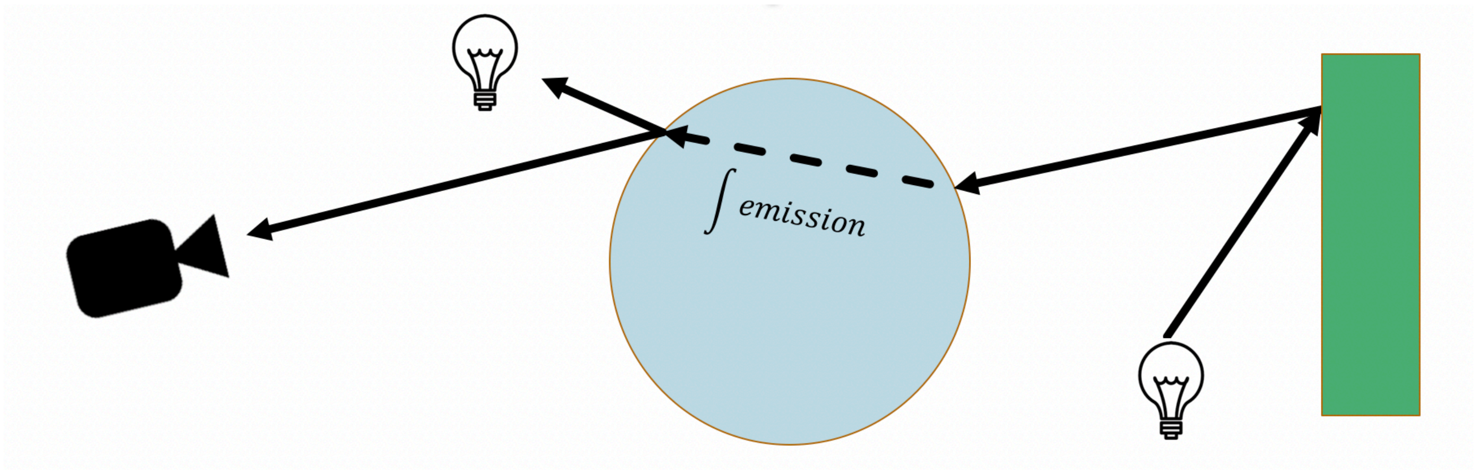
When the **VolumeLight** is sampled, the shape of the light is intersected and the emission from the **VolumeLight** is calculated by integrating the emission over the interior of the shape. To do this, a line segment through the shape must be determined; this segement appropximates the emission leaving the **VolumeLight** at the point and angle of intersection at which the ray sampled the light.

The figure below demonstrates the difference between a uniformly-lit **AreaLight** (left) and a uniformly-lit **VolumeLight** (right).



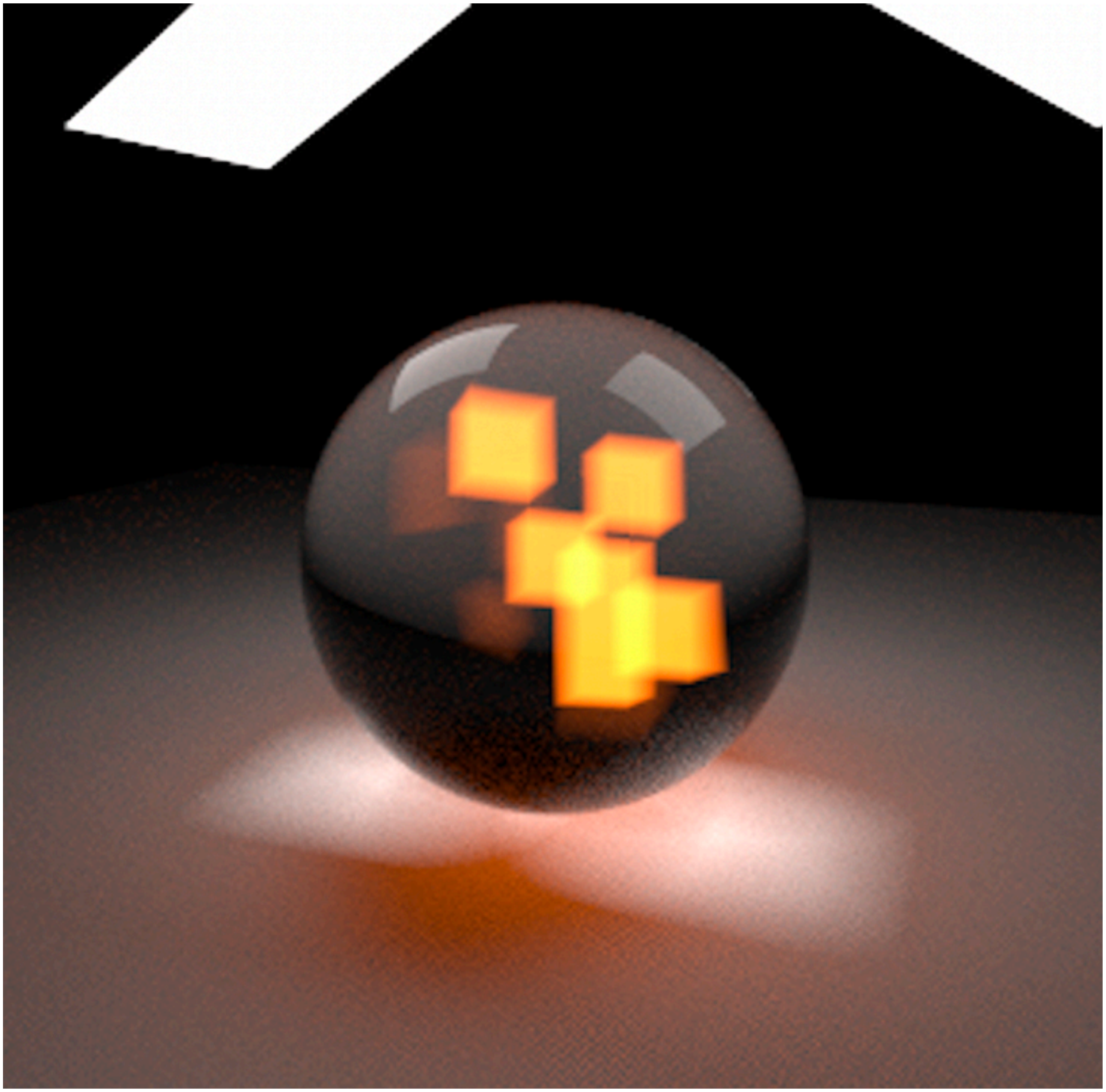
A significant advantage of implementing volumetric lighting this way is that light sources such as molten glass are sampled like ordinary lights; since they are recognized light sources, **VolumeLight**s are handled by **pbrt** in a very elegant way and can be sampled almost as efficiently as other light types.

One complication of this approach, however, is that the surface effects of the **VolumeLight** must be known at light-sampling time. Because of this, I made some substantial accomodations in the path integrator in **pbrt**. When the integrator samples a **VolumeLight**, it is necessary to know the exact intersection point, sampled BSDF angle, and refractive/transmissive properties of the **VolumeLight**'s surface; all of this information is critical for returning an accurate emission from the **VolumeLight**. With some careful modifications, I rearranged the order of steps taken in the path integrator to collect all of this information before **VolumeLight** sampling occurs. The figure below illustrates an emissive, refactive blue volume in which all of this information is used to compute the added volumetric emission along a refracted ray; also, notice the contribution of reflections off the the surface and the refracted transmission from the green wall behind the **VolumeLight**.



In my prototype of **VolumeLight**, the integration over the emissive volume is done using a midpoint-method numerical integral approximation; the line segment used is chosen using the point of intersection with the sampling ray and the refracted direction it takes within the surface.

To make my **VolumeLight** more versatile, I decided to implement a voxel grid which could be used to specify the intensity of the emission at arbitrary positions within the volume. I use tri-linear interpolation between the voxels and allow the specification of the voxel grid from a .csv file. The figure below demonstrates a 3x3x3 voxel grid of emissions inside a sphereical **VolumeLight**.



With these things in place, the final remaining challenge was to find a way to estimate the power of the **VolumeLight**. In actuality, when a volume light is declared in a **pbrt** scene file, the shape specified for the light is broken into many faces which each become individual **VolumeLight** class instances (they are related by their declaration, though). Therefore, all that is needed is to estimate the power of each face from the surface's mesh. In my prototype, for each face, I perform this estimation by sampling emissions the inward-facing hemisphere of the face and averaging the emissions along various random line segments originating from the face. Thus, the power of a given face is the expectation of its emission at a random viewing angle. With ~50 such samples random samples per face, my power estimation is sufficiently accurate to remove most of the variance that **VolumeLight**s would otherwise add to the scene. This power estimation's performance will degrade when voxel grids with higher variance are used. For molten glass, the emission is relatively slow-changing with respect to space.

Final Render

Once I had **VolumeLight** working correctly, I was ready to model and render my final scene. In this image, an artist is crafting a glass snail at a workbench. The voxel grid is used to make the bottom portion of the snail appear brighter than the shell.

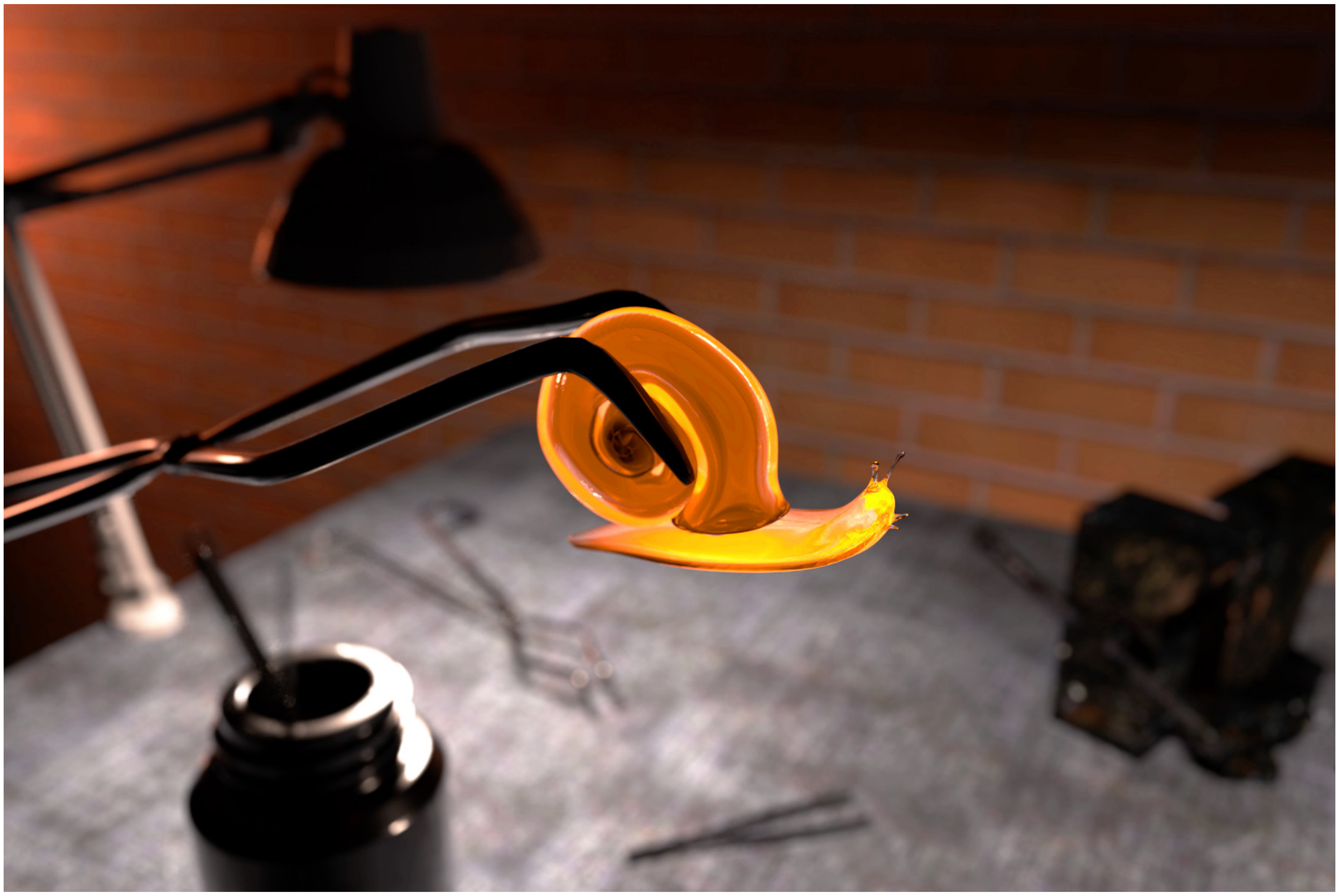


Image stats:

2880x1920 px, 2048 spp, ~11hr CPU render.

2.0GHz dual-core Intel Core i5, 4MB shared L3 cache, 8GB of 1866MHz LPDDR3 onboard memory. macOS Sierra.

~120K **VolumeLight** instances (120K faces of the snail).

All of the models used in the scene were purchased from TurboSquid.com.