Introduction

SolidWorks is a 3D mechanical CAD (computer-aided design) program that runs on Microsoft Windows. 3D models, designed in SolidWorks, can be displayed in photo-realistic views, but it requires additional processing by some special rendering software. One of such software products, available to the SolidWorks users, is PhotoView 360. PhotoView renders SolidWorks 3D models and saves the results into various popular graphic formats. It is reasonably fast and produces high quality views, however it does not support physically based rendering algorithm.

Physically based rendering provides images of much higher fidelity, but requires significant computational resources. That’s why traditionally it’s been outside the reach of the mainstream 3D mechanical CAD community and has been primarily used in digital art and entertainment industry. However, huge computational resources on the cloud platforms offer some interesting new opportunities to the CAD users.

This article reviews a SolidWorks add-in, which offers physically based rendering of 3D models on the Microsoft Windows Azure cloud platform.

Physically based rendering for SolidWorks

The SolidWorks add-in consists of two parts:

1. A converter from the SolidWorks data format into the format appropriate for the photo-realistic rendering - LuxRender, which is an open-source format
2. Rendering cluster that resides on the Windows Azure platform

Data Converter

Conversion of the SolidWorks model data into LuxRender format is performed locally on the user desktop. The format allows storing different portions of the 3D model data (geometry, light, camera, etc.) in separate files.

A successful conversion generates the following files:

- **Scene.lxs** – definition of the scene (contains definition of camera, lighting, render settings)
- **Data/luxMaterials.lxm** – definition of materials
- **Data/luxObjects.lxo** – definition of geometry

Current implementation is based on LuxRender version 0.8 and supports camera, geometry, lighting and materials data.

Figure 1.

Data flow from SolidWorks to the final render through the conversion into LuxRender format.

P. 1
Camera data

SolidWorks data is retrieved from the current active camera via ICamera and the viewport is calculated using IModelView APIs.

Table 1: Camera parameters

<table>
<thead>
<tr>
<th>SolidWorks</th>
<th>LuxRender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Camera.GetPosition</td>
<td>vector cameraPos</td>
</tr>
<tr>
<td>Camera.getViewVector</td>
<td>vector targetPos</td>
</tr>
<tr>
<td>Camera.GetUpVector</td>
<td>vector cameraUp</td>
</tr>
<tr>
<td>Camera.GetFocalDistance</td>
<td>FocalDistance</td>
</tr>
<tr>
<td>Camera.FieldOfViewAngle</td>
<td>Fov</td>
</tr>
<tr>
<td>Camera.AspectRatio</td>
<td>FrameAspectRatio</td>
</tr>
<tr>
<td>IModelView.GetVisibleBox</td>
<td>xresolution</td>
</tr>
<tr>
<td></td>
<td>yresolution</td>
</tr>
</tbody>
</table>

Geometry information

Data is extracted by traversing feature tree of the SolidWorks model using ModelDoc2.FeatureManager.GetFeatures API, selecting only features with bodies. Each body is tessellated by Body2.GetTessellation API, which returns the list of triangles (triangle indexes, positions, normals).

Appropriate material of each body was identified and associated with the corresponding triangles. Default material was used for the bodies that didn’t have an explicitly defined material.

Table 2: Body and material information

<table>
<thead>
<tr>
<th>SolidWorks</th>
<th>LuxRender</th>
</tr>
</thead>
<tbody>
<tr>
<td>Body2.GetTessellation</td>
<td>Shape “trianglemesh”</td>
</tr>
<tr>
<td></td>
<td>vector&lt;int&gt; indices</td>
</tr>
<tr>
<td></td>
<td>vector points</td>
</tr>
<tr>
<td></td>
<td>normal vector</td>
</tr>
</tbody>
</table>

Lighting

Data about single light sources is obtained by IModelDoc2.LightSourcePropertyValues API for each light source.

SolidWorks directional, spot and point lights have ambient component, but LuxRender lights don’t have it. So, these types of lights are converted into two light sources: light of the same type, plus ambient light.

See LuxRender lighting for detailed information about Lux light sources.

LuxRender supports “light group” feature. One or more lights can be combined in a light group. LuxRender renders the scene using different light groups to produce different images. It’s possible to change intensity and color of each light group without pausing the rendering.

Current version of the LuxRender add-in creates separate light group for each light source.
Materials

The list of materials from the SolidWorks model is retrieved by ModelDocExtension.GetRenderMaterials and geometry, associated with each material, accessed by IRenderMaterial.GetEntities API.

In the current implementation all materials are converted to “glossy” materials. Transparent objects and textures are supported too. Additional material properties can be obtained by parsing p2m material file.

See LuxRender materials for detailed information about materials in LuxRender.

Rendering Cluster on Windows Azure

Windows Azure offers computing resources, data storage and other cloud specific services like SQL Azure, Service Bus, Caching, etc.

For more details see Windows Azure Features.

Below is a schematic representation of the distribution of different tasks and operations of the rendering cluster on the Windows Azure resources:

- **Dispatcher**: processes rendering requests from the SolidWorks add-in, which includes allocating and freeing rendering nodes for each request, initializing rendering, returning intermediate and final results back to the add-in

- **Render**: renders SolidWorks models and saves intermediate results in the dedicated storage

- **Merger**: merges intermediate rendering results and converts it into an image file of the format specified in the initiating request

![Figure 2. Rendering on Windows Azure](image)

More details about rendering on Windows Azure can be found in the article “Adaptation of Image Processing Software for Windows Azure and HPC Cluster”.
Comparison of the rendering outputs from PhotoView 360 and SolidWorks add-in for physically based rendering on Microsoft Windows Azure

Two SolidWorks models were used for the comparison – a model of a ruby crystal (‘Ruby’) and a model of a glass of water (‘Water’). With these types of models it is easier to illustrate the advantages of the physically based algorithm, since light rays, passing through the glass or water, create numerous light flux refractions (especially inside crystalline solids), heterogeneous shadows and reflections, which are ideal for demonstrating the power of physically based render.

50 computing nodes were used for the rendering of each model for an hour. To show the light flux refractions inside ‘Ruby’ the model was rendered with the sources of light placed inside the model, and for a separate rendering an external light source was used.

‘Ruby’ with the internal light source

On the resulting images below you can clearly see the depth of the body, how the light plays inside the body, how the colored reflections fall on the table surface with the physically based rendering, while photo-realistic algorithm of PhotoView 360 only offers an imitation of these physical phenomena.

‘Ruby’ with the external light source

By comparing the two images below, you will notice on the output from the physically based render how the light is softly distributed, you can see the facets on the back side of the crystal, as well as the colored reflections. At the same time, on the image obtained from PhotoView the contrast between facets is coarser and the crystal is not transparent.
‘Water’ with the external light source

It’s hard not to notice the heterogeneous shadow falling from the glass with the light spot near the bottom; also a big light spot is present on the reflection of the glass itself. Creating such effects is only possible with physically based rendering, as they are hard to predict ahead of time.

![Figure 7. Physically based algorithm](image1)
![Figure 8. Photo-realistic algorithm](image2)

## Conclusion

Both, photo-realistic and physically-based rendering, algorithms have their advantages and limitations.

Photo-realistic algorithm is fast, the average time of rendering is within 2 minutes, however the resulting images miss some critical natural effects.

Physically-based algorithm demands significant computational resources, however it produces images comparable to what you may experience in real life.

Each algorithm serves its specific purpose - photo-realistic is good for visualization of intermediate results of CAD models and for non-critical presentations, while physically-based algorithm, like the one implemented in our add-in, is more appropriate for a ‘big stage’ type of a presentation, the demonstration to potential customers, or for the artistic renderings.

Taking advantage of a cloud platform, such as Microsoft Windows Azure, makes the physically-based rendering much more accessible for the mainstream CAD community, by providing scalable computing resources that are technically and financially challenging to set up on premises.

---

1. Current version of add-in does not support surfaces.
2. Environment light data, which can be retrieved from the scene p2s file, has not been implemented.