

# AUTHORING TOOL DESIGN DOCUMENT OUTLINE

**TITLE:** Stylit

**Developers:** Li Zheng, Keyi Yu

## PROJECT SUMMARY

---

Movies, games and advertisements require a bunch of artists and designers to create virtual characters and scenes. Before the development of computer graphics technology, artists can only do these works with hand drawing. Currently, artists can improve the work efficiency with the help of computer-aided design tools, which make movies and games with complex scenes and high-poly models possible. Realistic rendering technologies have been widely used. However, non-realistic rendering technologies like stylizing synthetic don't have many mature applications.

Our authoring tool is to give 3D models hand-crafted artistic appearances. The target audiences are artists, animators and game designers, especially those work on independent games and animations. Pursuing of realistic rendering of movies and games can meet the problem of homogenization. Stylized rendering can help movies and games stand out from others. Traditionally, to create a non-realistic scene, the artists need to make specific textures for the models, which is a great cost for independent developers. Our authoring tool can significantly reduce the workload of artists and developers. They just need to provide an exemplar painting on a "sphere on the table" scene. For the test version of the authoring tool, we will provide some sample styles. Then, for a more complex target scene with a similar lighting environment, the tool computes the light propagation and uses it to guide the synthesis. The target scene will exhibit a similar visual style to the exemplar.

We plan to develop the authoring tool as a Maya Plugin. It is based on the paper *Stylit: illumination-guided example-based stylization of 3d renderings* [Fischer et al. 2016]. It combines the Light Path Expressions (LPEs) with image analogies to implement style synthesization. We will develop a Cuda test app to start, then develop C++ plugin. Next step is strictly MEL or Python scripting. Finally, C++ and Maya will be deeply integrated.

## 1. AUTHORING TOOL DESIGN

---

### 1.1. Significance of Problem or Production/Development Need

Physical based rendering technology has produced many realistic games and movies. To distinguish a game or movies from others, the producer pays more attention to the stylization. So a tool that can efficiently synthesize the design of artists and 3D models is required. We plan to develop an authoring tool that can interactively reflect the designed visual effect to the model. the tool will provide an artistic visual effect for the digital products with very lost cost, especially suitable for independent developers.

### 1.2. Technology

The paper we choose is *Stylit: illumination-guided example-based stylization of 3d renderings* [Fišer et al. 2016]. It proposes a stylization technology combining Light Path Expressions (LPEs) and image analogies.

**LPE:** It uses a three-pass, bidirectional ray tracing algorithm that traces rays from both the lights and the eye. 1) The “size pass” records visibility information on diffuse surfaces; 2) the “light pass” progressively traces rays from lights and bright surfaces to deposit photons on diffuse surfaces to construct the radiosity textures; 3) and the “eye pass” traces rays from the eye, collecting light from diffuse surfaces. One of its uses is to separate the various illumination effects into distinct buffers for the purposes of filtering or in order to use a different rendering algorithm for each.

**Image Analogies:** A pair of images serves as an exemplar. For each pixel in the target, the algorithm finds the best corresponding location in the unfiltered source and transfers the look from the filtered counterpart.

We choose this paper because it comprehensively reviews the previous stylization technologies and innovatively takes advantage of the light propagation to improve the stylization performance.

### 1.3. Design Goals

Describe how your authoring tool will address the problem or need

#### 1.3.1 Target Audience.

Artists, animators and game designers can use this tool to interactively design the visual effects of their works. It’s especially useful for independent game and movie producers.

#### 1.3.2 User goals and objectives

Users can use this tool to give the 3D models a visual effect corresponding to the exemplar they provide.

#### 1.3.3 Tool features and functionality

The tool provides several visual effect samples that can be used to render user provided models. Also, users can use the exemplar they designed.

#### 1.3.4 Tool input and output



“sphere on the table” exemplars

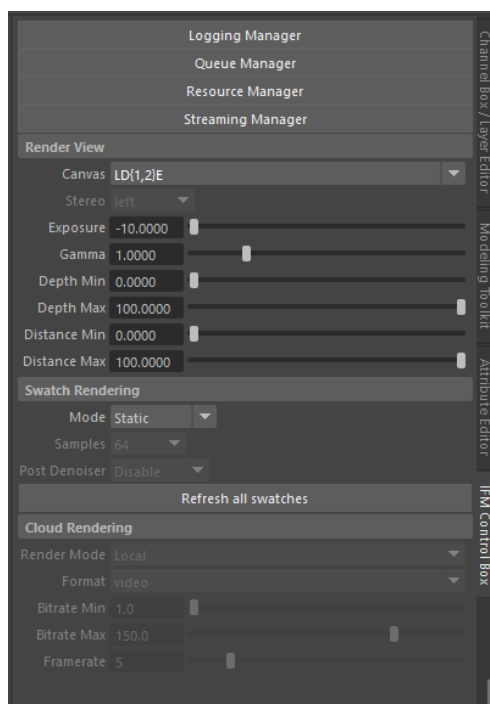
**Input:** The user should provide the model they want to render. It’s better to put the model in our provided lighting environment, which will have the best

performance. Alos, the users need to provide a “sphere on the table” exemplar, as shown in the figure above. We also have preset exemplars. For the test version, the users should provide four LEP images for rendered models. Users can achieve them from the Arnold or Iray renderer. The images should have JPG or PNG format.

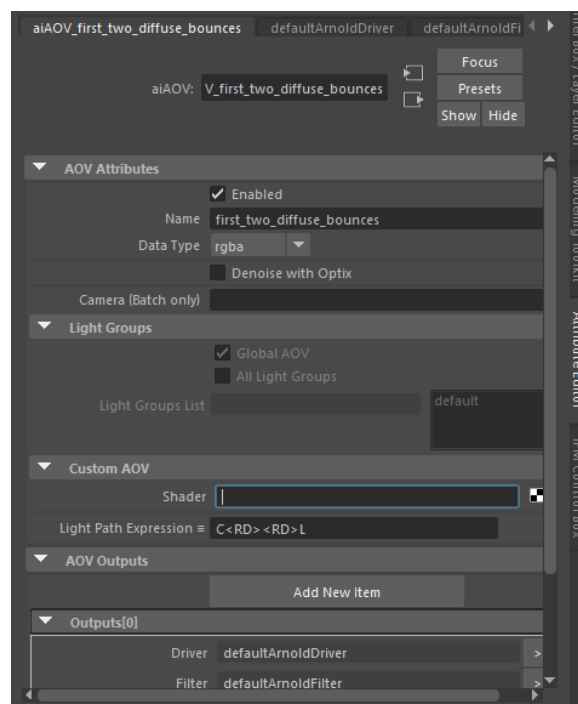
**Output:** For the test version, the output will be a stylized image with JPG or PNG format. The images can be downloaded to the folder and displayed in Maya’s renderView as well. Ideally, we want to get an real-time and interactive output from a renderer like Arnold or Iray.

## 1.4. User Interface

### 1.4.1 GUI Components and Layout



Iray IFM Control Box

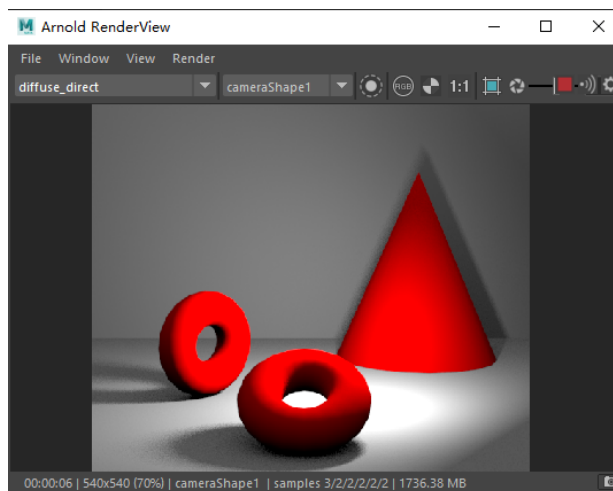


Arnold AOV Attribute Editor

Users can use Iray or Arnold to set the LPEs. Maya installs the Arnold render in default. If you want to use the Iray plugin, it can be downloaded from <https://www.irayplugins.com/iray-for-maya/>.

The LPEs can be set in Render settings. They are a little different for Arnold and Iray. Please refer to <https://docs.arnoldrenderer.com/display/A5ARP/Light+Path+Expression+AOVs> and

<https://raytracing-docs.nvidia.com/iray/manual/index.html#reference#light-path-expressions>.



Then you can save images in JPG or PNG format from the render view of Maya or Arnold.

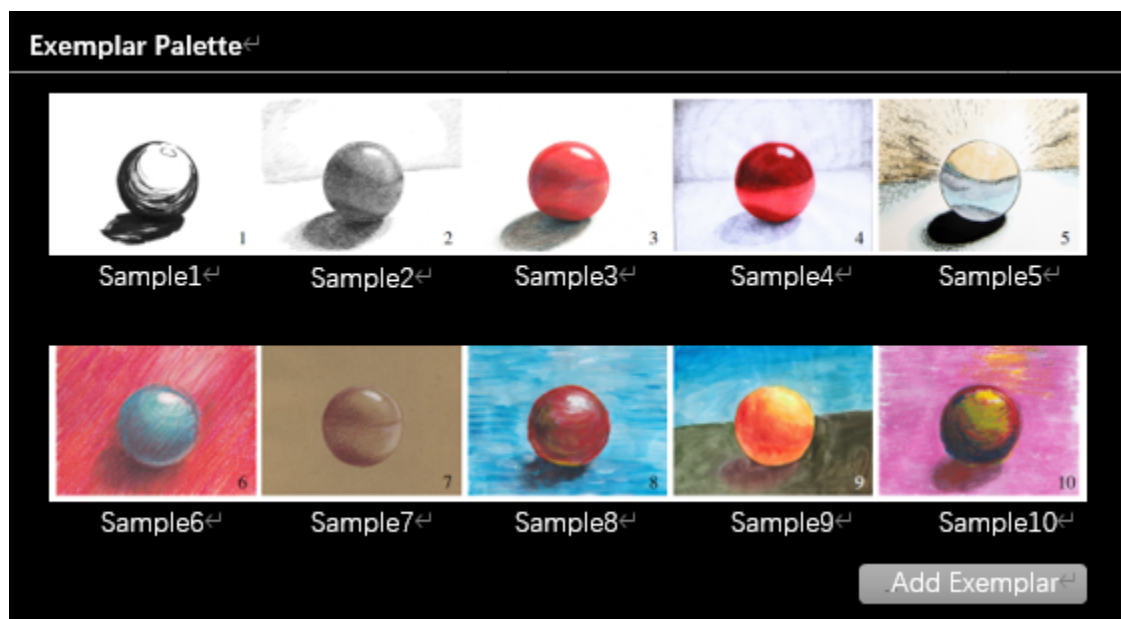


This is the **Stylit** GUI, it is the main window of the authoring tool. Users can open it from Maya's menu.

**Load LPEs:** For the test version, the user should provide four light path expressions of the rendering target, including direct diffuse, direct specular, first two diffuse bounces and diffuse interreflection. they can be achieved from renderers like Arnold and Iray.

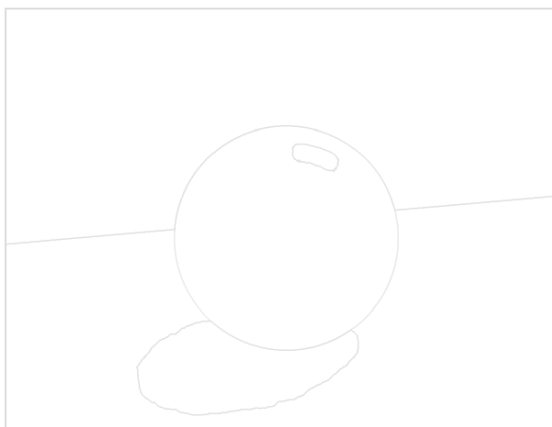
**Select exemplar:** the user can load an exemplar from the **Exemplar Palette**.

**Generate:** Generate the stylized result.

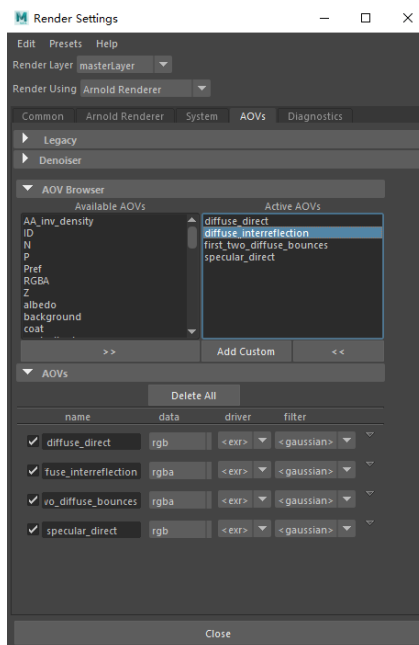


This is the **Exemplar Palette** GUI. It can be opened from the **Stylit** GUI. Double click an image, it will be loaded as an exemplar. Users can also load their custom exemplar by clicking on the **Add Exemplar** button.

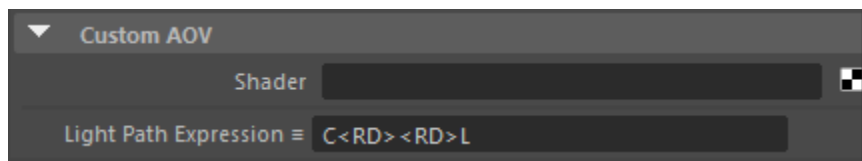
### 1.4.2 User Tasks



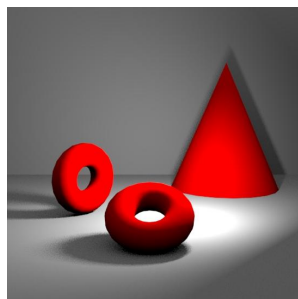
The user needs to draw on the provided “ball on the table” scene, where the highlight and shadow areas are outlined. Also, they need some basic knowledge about renderers (Arnold, Iray, etc.) and be able to generate LPE images with them. Take Arnold as an example, the user should follow these steps:



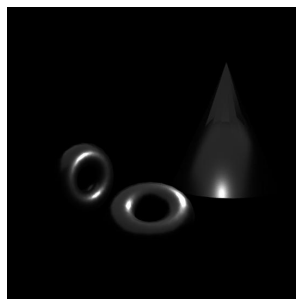
1. Add AOVs in Render Setting. You can select diffuse\_direct and specular\_direct AOV from available AOVs. Then add another two custom AOVs.



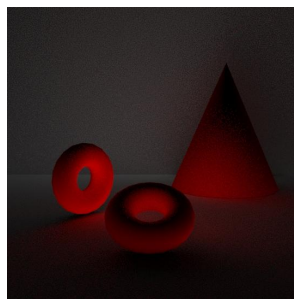
2. Type in the light path expression. For Arnold, the first two diffuse bounces expression is  $C<RD><RD>L$ . The interreflection expression is  $C<RD><RD>.*L$ . For Iray, they should be  $LD\{1,2\}E$  and  $L.*DDE$ . Direct diffuse is  $LDE$ . Direct specular is  $LSE$ .



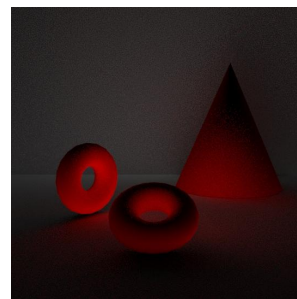
Direct Diffuse



Direct Specular



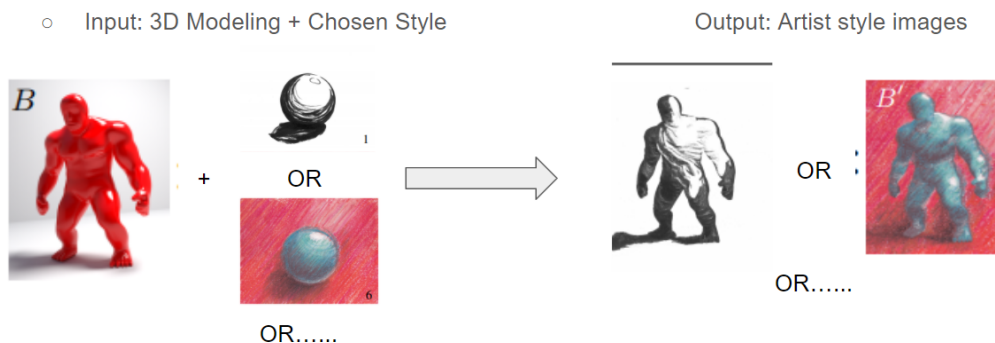
first two diffuse bounces



interreflection

3. Save four LPE images in JPG or PNG format from the render view.

### 1.4.3 Work Flow



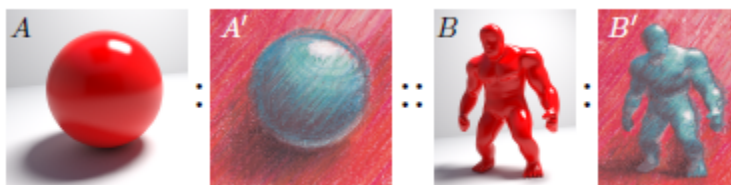
1. Generate four LPEs, including direct diffuse, direct specular, first two diffuse bounces and diffuse interreflection, with a renderder like Arnold or Iray.
2. Select an exemplar from file or the preset samples.
3. Click the Generate button.

The first step is just for a test version. Ideally, this step will be integrated to our plugin with Iray API. Currently, we haven't dive into the API, so this step is put here just in case.

## 2. AUTHORIZING TOOL DEVELOPMENT

### 2.1. Technical Approach

#### 2.1.1 Algorithm Details



Given three multi-channel images  $A$  (exemplar scene rendered with LPE channels),  $A'$  (stylized exemplar aligned to the exemplar scene), and  $B$  (target scene rendered with LPE channels). The authoring tool is to synthesize a new target image  $B'$  such that  $A : A' :: B : B'$ .

For each resolution level and each pixel  $q \in B'$  in scan-line order, a best matching pixel  $p$  is found in the source  $A'$  such that

$$E(A, B, p, q, \mu) = \|A'(p) - b'(q)\|^2 + \mu \|A(p) - b(q)\|^2 \quad (1)$$

is minimized. Here  $A = \{A, A'\}$ ,  $B = \{B, B'\}$ , and  $\mu$  is a weight that controls the influence of guidance. For any image  $I$ , we use  $I(p)$  to denote a feature vector at a pixel  $p$ . The vector  $I(p)$  is a concatenation of all pixels in a small square patch of width  $w$  centered at the pixel  $p$ , where each pixel can have multiple feature channels, containing colors of the full rendered image (RGB) and four LPE channels (each stored as a RGB image):

$$\{A, B\} = (FULL, LDE, LSE, L.*DDE, LD\{1,2\}E) \quad (2)$$

FULL represents the full global illumination render. LDE is direct diffuse. LSE is direct specular,  $L.*DDE$  is first two diffuse bounces.  $LD\{1,2\}E$  is diffuse interreflection. An optimization scheme is to minimize the following energy:

$$\sum_{q \in B} \min_{p \in A} E(A, B, p, q, \mu) \quad (3)$$

using EM-like iteration executed multiple times from coarse to fine resolution:

```

input : multi-channel images  $A = \{A, A'\}$  and  $B_k = \{B, B'_k\}$ 
output: synthesized target image  $B'_{k+1}$ 
for each pixel  $q \in B_k$  do
   $NNF(q) = \underset{p \in A}{\operatorname{argmin}} E(A, B_k, p, q, \mu)$ 
for each pixel  $q \in B_k$  do
   $B'_{k+1}(q) = \operatorname{Average}(A, NNF, q)$ 

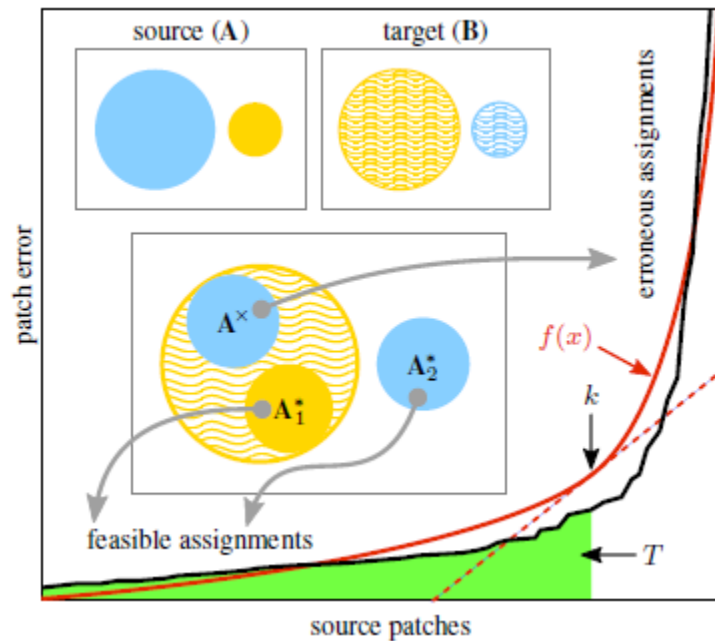
```

**Algorithm 1:** EM-like iteration used to minimize energy (3).

Here  $B'_k$  denotes the current pixel values stored in  $B_k$  and  $B'_{k+1}$  are the updated values. NNF is the nearest neighbour field that assigns source patch to each target patch and function *Average* computes the average color of colocated pixels in neighbour patches.

The uniform source patch usage can alleviate the wash-out effect. However, this restriction is suitable only when each randomly picked sub-region of the source texture is perceived similarly. Enforcing uniform patch usage in our scenario would create artifacts. Our solution is based on the idea of reversed NNF retrieval [Rosenberger et al. 2009; Jamriřska et al. 2015], in which a best matching target patch is retrieved for each source patch.





It is observed that, in practice, when we sort all matching error values and plot them with normalized axes, the resulting graph has a distinct, hyperbolic shape. The graph can be fitted to a hyperbolic function  $f(x) = (a - bx)^{-1}$ . Retrieve the point where  $f'(x) = 1$ , i.e.,  $k = \sqrt{1/b} + a/b$ . Patches with indices above  $k$  are probably erroneous assignments that need to be avoided. We thus set a feasible error budget  $T$  that is an integral of all patch errors with indices below  $k$  and modify the original source-to-target assignment in a way that maximizes the number of used source patches  $|A^*|$  while satisfying an additional feasibility constraint:

$$\sum_{p \in A^*} \min_{q \in B} E(A^*, B, p, q, \mu) < T \quad (4)$$

We can repeat the retrieval and reuse good source patches to cover remaining positions in the target. This iterative scheme stops when all target patches are covered.

### 2.1.2 Maya or Houdini Interface and Integration

We will implement the algorithms in Maya. A dialog UI component will be created with MEL to access the C++ based plugin. A renderer UI will be used to show the stylized rendering result.

**MfnMesh**: Polygonal surface function set

**MViewportRenderer**: A class which represents a hardware viewport

**MImage**: A class provides methods for reading file images stored on disk.

### 2.1.3 Software Design and Development

- a. The stylized synthetic process will be implemented in C++, including these class:

**Image Class**: A class responsible for reading from images and writing to images

```
class image {
private:
    int xSize;
    int ySize;
    glm::vec3* pixels;

public:
    image(int x, int y);
    ~image();
    void readFromFile(std::string filename);
    void setPixel(int x, int y, const glm::vec3& pixel);
    void savePNG(const std::string& baseFilename);
    void saveHDR(const std::string& baseFilename);
};
```

**FeatureVector Class**: The vector  $I(p)$  is a concatenation of all pixels in a small square patch of width  $w$  centered at the pixel  $p$ , where each pixel contains colors of the full rendered image (RGB) and four LPE channels (each stored as a RGB image).

```
class FeatureVector {
public:
    std::unique_ptr<image> RGB;
    std::unique_ptr<image> LDE;
    std::unique_ptr<image> LSE;
    std::unique_ptr<image> LDDE;
    std::unique_ptr<image> LD12E;

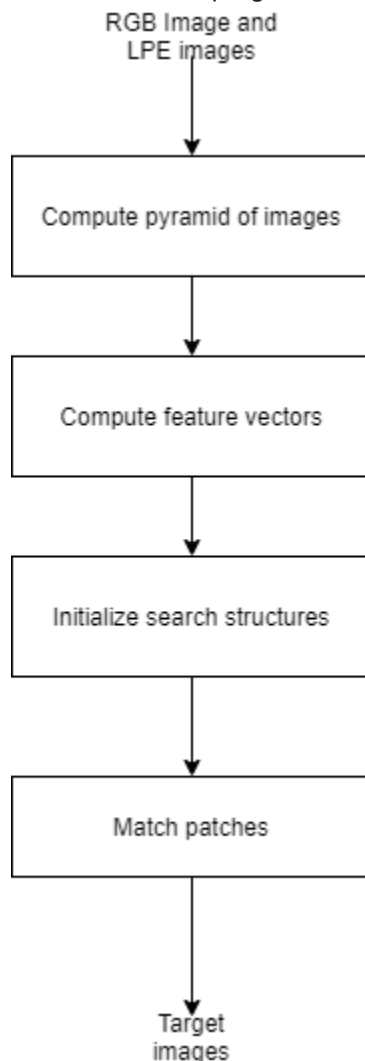
    int level;

    FeatureVector(std::unique_ptr<image> rgb,
                  std::unique_ptr<image> lde,
                  std::unique_ptr<image> lse,
                  std::unique_ptr<image> ldde,
                  std::unique_ptr<image> ld12e);
    ~FeatureVector();
};
```

**Pyramid Class**: A class containing all the FeatureVectors at different levels.

```
class Pyramid {  
public:  
    int levels;  
  
    std::vector<FeatureVector> featureAtAllLevels;  
};
```

- b. Detailed breakdown of the associated program structure



- c. A standalone test app

Firstly we will generate RGB and LPE images in Maya and save them in a folder. Then we will run our main algorithm including reading from those images and matching. Finally we will save the result images in the same folder and check the accuracy of the result.

- d. Thirdparties

Our pipeline needs to load a Iray plugin which helps the users to generate RGB and LPE images.

## **2.2. Target Platforms**

### **2.2.1 Hardware**

CPU: Any modern CPU

Graphics card: A Nvidia card is required

### **2.2.2 Software**

Version of Windows: Windows 10

Maya: 2019 or 2020

IRay

## **2.3. Software Versions**

### **2.3.1 Alpha Version Features (first prototype)**

- Complete set of features to be included in the alpha version.
  - Output an image in exemplar style outside of Maya
- Important development milestones.
  - Week1:
    - Learning how to generate rendering images guided by LPEs with Arnold Plugin
    - Preparing source images and cartoon images
  - Week2:
    - Setting up a basic image filtering program
  - Week3.4:
    - Implementing the main algorithm
- Demo/test app to show off the alpha features.
  - Loading images rendered by Iray to our C++ program and saving the result image in a folder

### **2.3.2 Beta Version Features**

- Complete set of features to be included in the beta version.
  - Output an image in exemplar style in Maya
- Important development milestones.
  - Week1:
    - Implementing part UI and Node
  - Week2:

- Integrating our plugin with Iray
- Demo/test app to create to show off the beta features.
  - Loading images to our plugin and compare the output with the results in paper

### 2.3.3 Description of any demos or tutorials

- How will users know how to use your tool?
  - We will provide a demo and a tutorial on how to use our authoring tool.
- Describe the demos and/or tutorials you plan to develop that will ship with the final version.
  - A demo will be provided to show how to use our authoring tool step by step.
  - A tutorial will be provided to show how to use our authoring tool step by step.

## 3. WORK PLAN

---

### 3.1. Tasks and Subtasks

<b>Task 1 – Arnold LPEs</b>
-----------------------------

<b>Duration: 03/02/2021 - 03/07/2021(5 days)</b>
--

- **Subtask 1.1. LPE Concept**  
*LPE is a way to store light information. We need to learn the basic concept of LPE.*
- **Subtask 1.2. How to select LPEs**  
*In the paper, they select several kinds of LPEs for image analogies. We need to dive into the paper and decide what kind of LPE images we need for our authoring tool*
- **Subtask 1.3. LPE in Arnold**  
*We will watch tutorials online on how to generate rendering images guided with LPEs in Arnold.*

<b>Task 2 – Preparing sources images and exemplars</b>
--

<b>Duration: 03/02/2021 - 03/07/2021(5 days)</b>
--

- **Subtask 2.1. Sources images**  
*We will follow this paper and generate images which render the sphere on the desk.*

## CIS660: Advanced Topics in Computer Graphics and Animation

- **Subtask 2.2. Exemplar**

*In this paper, five trained artists draw the exemplars which need for image analogies. We will look for those exemplars online and find more exemplars.*

- **Subtask 2.3. Light information**

*In week1, we get those rendering images guided by LPEs. We need to get light information from channels in those images.*

### Task 3 – Setting up project

Duration: 03/08/2021 - 03/14/2021(6 days)

- **Subtask 3.1. Setup c++ project 2 days Keyi Yu**

*We will download the OpenGL library and std\_image library online. Then we will create a new project in Visual Studio. Next we will include those libraries in properties and make sure we can include them in our project.*

- **Subtask 3.2. Image class 1 day Keyi Yu**

*All the features including RGB and LPE are stored in images, so we use an Images class to store those values.*

- **Subtask 3.3. FeatureVector Class 2 day Li Zheng**

*We use a FeatureVector class to store RGB and LPE features which are read from images.*

### Task 4 – Image Analogies(Main algorithm)

Duration: 03/15/2021 - 03/29/2021(13 days)

- **Subtask 4.1. Computing pyramids and feature vectors 7 days Li Zheng**

*Firstly, we need to Compute Gaussian pyramids for  $A'$ ,  $A$ , and  $B$ (three input images). We also need to extract feature vectors from them. In this paper, the authors use FULL, LDE, LSE,  $L * DDE$ ,  $LD\{1,2\}E$  in LPEs as feature vectors.*

- **Subtask 4.2. BestMatch Function 7 days Keyi Yu**

*The heart of the image analogies algorithm is the BESTMATCH subroutine. We will implement this function to find the best match pixel in the source images for target images.*

- **Subtask 4.3. Integrate functions and test results 6 days Li Zheng&Keyi Yu**

## CIS660: Advanced Topics in Computer Graphics and Animation

*After finding the best match pixels, we can apply the filter to the source image and get the final result. We will simply compare the style of our results to the exemplar with our eyes.*

### Task 5 – MEL menu and UI

Duration: 03/29/2021 - 04/04/2021(7 days)

- **Subtask 5.1. Menu and UI**      4 days Li Zheng  
*We will write MEL codes to generate the menu and UI mentioned in 1.4.1.*
- **Subtask 5.2. Node**      4 days Keyi Yu  
*We will use c++ to create Node class as we did in previous assignments for input and output images.*
- **Subtask 5.3. Display output images**      3 days Li Zheng  
*Besides saving output images in folders, our authoring tools will also display images in Maya*

### Task 6 – Integrate Arnold/Iray with our Plugin

Duration: 04/05/2021 - 04/27/2021(22 days)

- **Subtask 6.1. Arnold**      5 days Keyi Yu  
*We hope to integrate the rendering engine with our plugin. We will learn both two SDK- Arnold and Iray. Finally, we will decide on either of them for our authoring tool.*
- **Subtask 6.2. IRay**      5 days Li Zheng  
*We hope to integrate the rendering engine with our plugin. We will learn both two SDK- Arnold and Iray. Finally, we will decide on either of them for our authoring tool.*
- **Subtask 6.3. Modify our plugin to integrate rendering engine** 17 days Keyi Yu&Li Zheng  
*Once we are able to call rendering engine apis in our plugin, we could make changes to our code in that users can directly generate artist-style images from 3D models.*

### Task 7 – Demo/Tutorial

Duration: 04/28/2021 - 05/10/2021(12 days)

- **Subtask 7.1. Demo**      **7 days Keyi Yu**

*We will make a recording of our plugin showing how to use our plugin step by step.*

- **Subtask 7.2. Tutorial**      **7 days Li Zheng**

*We will write a document in detail to illustrate how to use our plugin step by step. The document will include libraries, dependencies, environment and instructions of using our plugin.*

- **Subtask 7.3. Package our code**      **1 days Keyi Yu&Li Zheng**

*We will clean our code and comments importance procedures and make sure the users can load our plugin easily.*

## 3.2. Milestones

### 3.2.1 Alpha Version

Task1, Task2

Task3

Task4

### 3.2.2 Beta Version

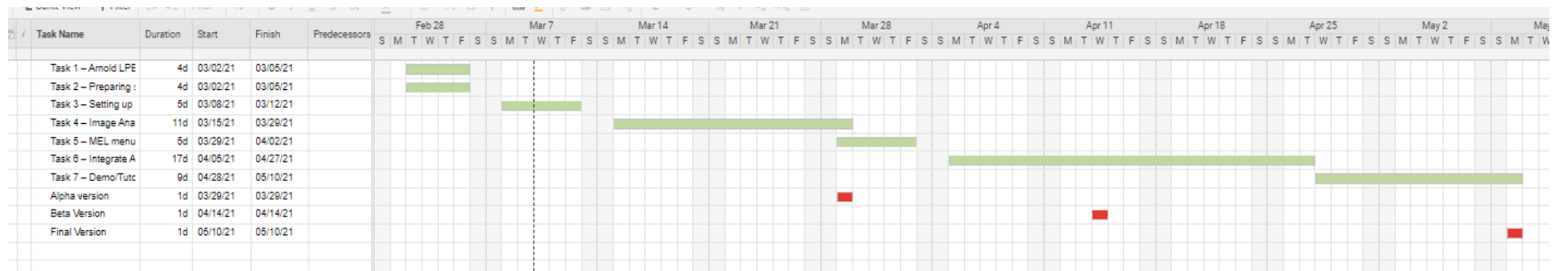
Task5, Task6.1, Task6.2

### 3.2.3 Final Version

Task7

## 3.3. Schedule





## **4. RELATED RESEARCH**

---

### **Introduction**

[haeberli1990] is a pioneer to convert a photo or computer generated image into digital painting. Many techniques were developed to achieve this goal, including physical simulation, procedural techniques, advanced image filtering and algorithmic composition of exemplar strokes.

Previous example-based approaches [Sloan2001; Hertzmann2001; B´enard2013; Fiřer2014; Barnes2015] have made significant progress, but the synthesized results still have a distinctively synthetic look when compared to real artwork. The first limiting factor is that these techniques rely mainly on color information, but actual artists pay as much attention to lighting effects when painting a scene. Although normals can partially alleviate this limitation [Sloan2001], they are useful only for a simple shading scenario. The second limiting factor is that previous example-based stylization techniques can produce many artifacts [Sloan2001; Hertzmann2001; B´enard2013; Fiřer2014; Barnes2015; Kaspar2015; Jamriřska2015].

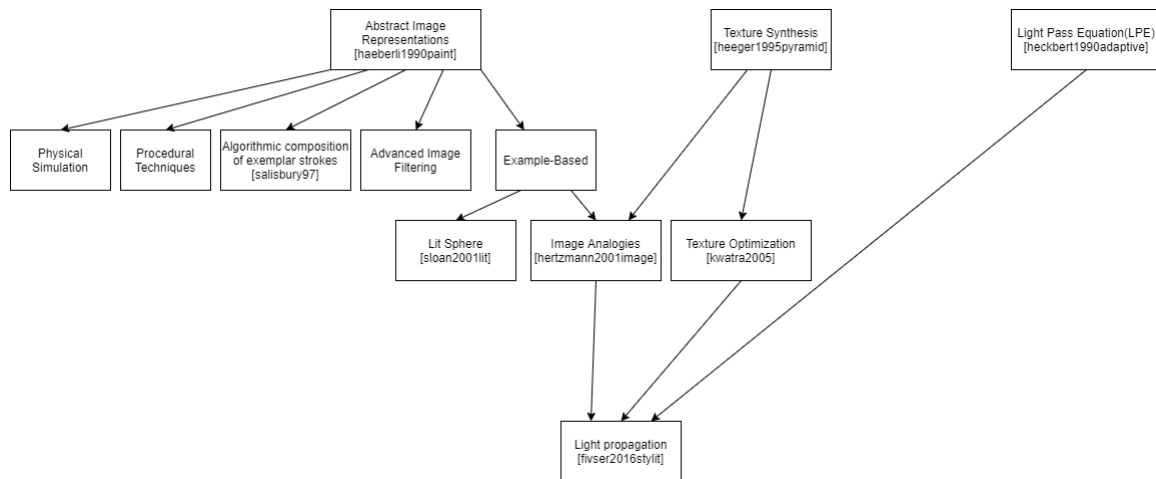
The authoring tool is based on a paper [Fiřer2016] that proposes a solution to alleviate these problems. It computes light propagation in a simple scene. Then uses it to guide the synthesis, which depicts various global illumination effects in arbitrary styles. To alleviate the artifacts, it uses source patches uniformly while controlling the overall error budget.

In this literature survey we trace from the paper [haeberli1990], diving into the example-based

Techniques including the lit sphere [Sloan2001] and image pair [Hertzmann2001]. [Hertzmann2001] integrated non-realistic rendering with texture synthesis, which could trace back to the paper [heeger1995pyramid]. Then, the original Hertzmann et al. synthesis algorithm was replaced by a texture optimization technique [Kwatra et al. 2005]. By adding the Light Path Expressions (LPEs) [hertzmann2001image] channels,

[Fīser2016] alleviates the artifacts and improves the fidelity of the synthesized image in global illumination scenarios.

## Graph



## **Content**

### **[haeberli1990paint]:**

In this seminal paper, the author used an ordered list of strokes to represent images. Each stroke of the brush is described by a collection of attributes including position of the brush stroke, the RGB and Alpha color of the stroke, the size of the stroke, angle of the stroke in the painting and the shape of the brush stroke. This paper also described how to control those attributes in their program. In addition, it is easy to apply some operations on the paintings, like interpolation and extrapolation beyond two paintings and animation.

There are also some advanced painting techniques. Firstly, Interesting effects can be created by using arbitrary images to control the brush direction across the canvas. Another interesting technique is to blend the brush strokes onto the canvas.

### **[salisbury97]:**

In this paper, they introduced the notion of “orientable textures” and show how they can be used to readily convert 3D information in an image-based system for pen-and-ink illustration. In their interactive system, a user creates an illustration from a reference image by specifying three components: 1) : a greyscale target image that defines the desired tone at every point in the illustration; 2) a direction field that defines the desired orientation of texture at every point; 3) a stroke example set, or set of strokes, to fill in the tone areas.

### **[sloan2001lit]:**

This paper introduced The Lit Sphere-a generic example-based technique that uses a shaded sphere painted by an artist as the style exemplar. Pixels from this spherical exemplar are then transferred to the target 3D model using environment mapping, i.e, the color for a target pixel is transferred from the location in the source with the same normal.

**[hertzmann2001image]:**

This paper proposed a concept of image analogies where a pair of images serves as an exemplar. For each pixel in the target, the algorithm finds the best corresponding location in the unfiltered source and transfers the look from the filtered counterpart.

**[heeger1995pyramid]:**

This paper presents a technique for synthesizing an image that matches the appearance of a given texture sample. This technique works entirely from the example texture, requiring no additional information or adjustment. It starts with a digitized image and analyzes it to compute a number of texture parameter values. Those parameters are then used to synthesize a new texture image that looks like the original image.

**[Kwatra2005]:**

In contrast to most example-based techniques that do region-growing, this paper proposes a joint optimization approach that progressively refines the entire texture. Also, it is ideally suited to allow for controllable synthesis of textures. It defines a Markov Random Field (MRF)-based similarity metric for measuring the quality of synthesized texture with respect to a given input sample, which allows formulating the synthesis problem as minimization of an energy function.

**[Fišer2016]**

This paper presented an approach to example-based stylization of 3D renderings that takes into account illumination effects. It computes the light propagation and

uses it to guide the synthesis. Artists just need to provide an exemplar painting on a “sphere on the table” scene. Then, a more complex target scene with a similar lighting environment will exhibit similar visual style to the exemplar. The demonstration shows that this technique can handle a great variety of different stylizations.

**[heckbert1990adaptive]:**

This paper presents a rendering method designed to provide accurate, general simulation of global illumination for realistic image synthesis.

They use a three-pass, bidirectional ray tracing algorithm that traces rays from both the lights and the eye. 1) The “size pass” records visibility information on diffuse surfaces; 2) the “light pass” progressively traces rays from lights and bright surfaces to deposit photons on diffuse surfaces to construct the radiosity textures; 3) and the “eye pass” traces rays from the eye, collecting light from diffuse surfaces.

## References

- [haeberli1990paint] HAEBERLI, P. 1990. Paint by numbers: Abstract image representations. SIGGRAPH Computer Graphics 24, 4, 207–214.
- [salisbury97] SALISBURY, M. P., WONG, M. T., HUGHES, J. F., AND SALESIN, D. H. 1997. Orientable textures for image-based pen-and-ink illustration. In SIGGRAPH Conference Proceedings, 401–406.
- [sloan2001lit] SLOAN, P.-P. J., MARTIN, W., GOOCH, A., AND GOOCH, B. 2001. The Lit Sphere: A model for capturing NPR shading from art. In Proceedings of Graphics Interface, 143–150.

- [hertzmann2001image] HERTZMANN, A., JACOBS, C. E., OLIVER, N.,  
CURLESS, B.,  
AND SALESIN, D. H. 2001. Image analogies. In  
SIGGRAPH  
Conference Proceedings, 327–340.
- [Kwatra2005] KWATRA, V., ESSA, I. A., BOBICK, A. F., AND KWATRA, N. 2005.  
Texture optimization for example-based synthesis. ACM  
Transactions on Graphics 24, 3, 795–802.
- [heeger1995pyramid] Heeger, David J., and James R. Bergen. "Pyramid-based texture  
analysis/synthesis." Proceedings of the 22nd annual conference  
on  
Computer graphics and interactive techniques. 1995.
- [Fišer2016] Fišer, J., Jamriška, O., Lukáč, M., Shechtman, E., Asente, P., Lu,  
J., &  
Sýkora, D. (2016). Stylit: illumination-guided example-based  
stylization of  
3d renderings. ACM Transactions on Graphics (TOG), 35(4), 1-11.
- [hertzmann2001image] HECKBERT, P. S. 1990. Adaptive radiosity textures for  
bidirectional ray tracing. SIGGRAPH Computer  
Graphics 24,  
4, 145–154.
- [B'enard2013] B'ENARD, P., COLE, F., KASS, M., MORDATCH, I., HEGARTY,  
J., SENN, M. S., FLEISCHER, K., PESARE, D., AND BREEDEN,  
K. 2013. Stylizing animation by example. ACM Transactions  
on Graphics 32, 4, 119.

- [Fišer2014] FIŠER, J., LUKÁČ, M., JAMRIŠKA, O., ČADÍK, M., GINGOLD, Y., ASENTE, P., AND ŠYKORA, D. 2014. Color Me Noisy: Example-based rendering of hand-colored animations with temporal noise control. *Computer Graphics Forum* 33, 4, 1–10.
- [Barnes2015] BARNES, C., ZHANG, F.-L., LOU, L., WU, X., AND HU, S.-M. 2015. PatchTable: Efficient patch queries for large datasets and applications. *ACM Transactions on Graphics* 34, 4, 97.
- [Kaspar2015] KASPAR, A., NEUBERT, B., LISCHINSKI, D., PAULY, M., AND KOPF, J. 2015. Self tuning texture optimization. *Computer Graphics Forum* 34, 2, 349–360.
- [Jamriška2015] JAMRIŠKA, O., FIŠER, J., ASENTE, P., LU, J., SHECHTMAN, E., AND ŠYKORA, D. 2015. LazyFluids: Appearance transfer for fluid animations. *ACM Transactions on Graphics* 34, 4, 92.
- [Barnes2009] BARNES, C., SHECHTMAN, E., FINKELSTEIN, A., AND GOLDMAN, D. B. 2009. PatchMatch: A randomized correspondence algorithm for structural image editing. *ACM Transactions on Graphics* 28, 3, 24.

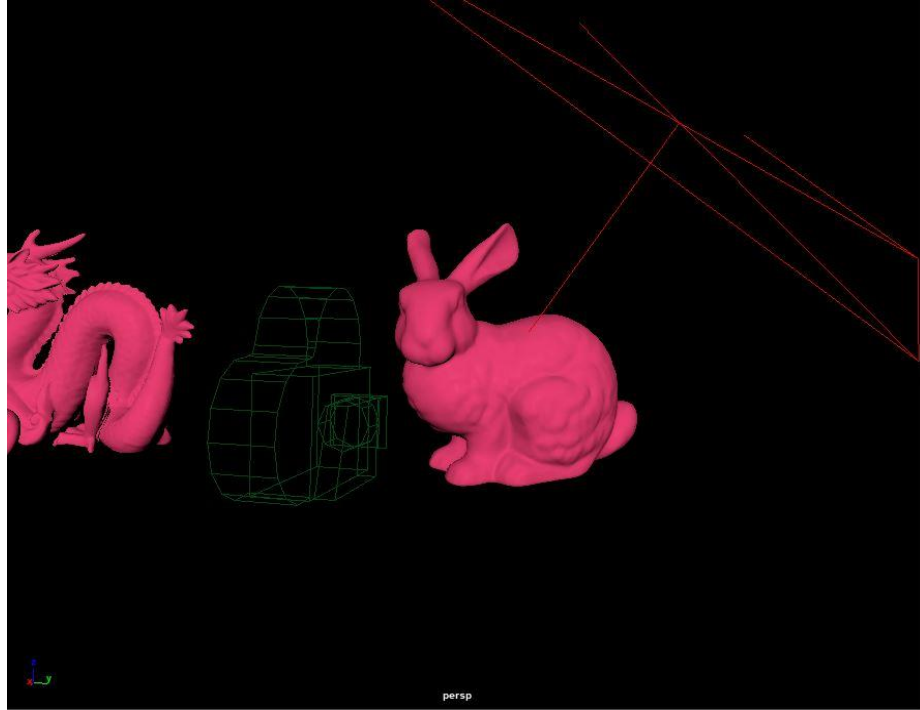


## 5. RESULTS

---

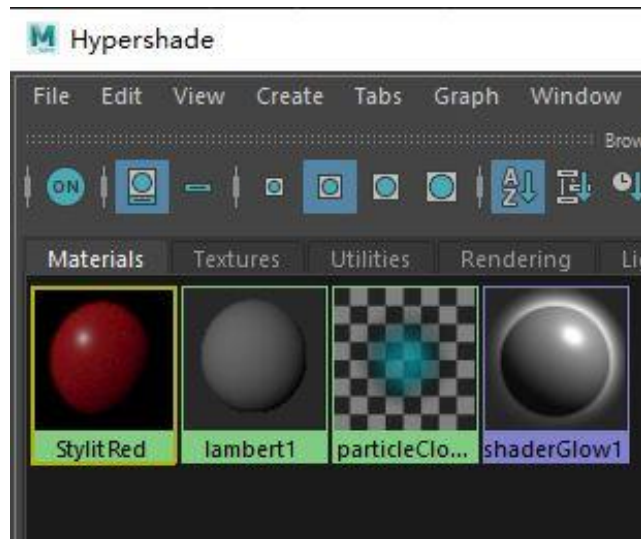
### STEP1: Gnererate LPEs

1. Open the default 'scene.mb'
2. : Load your custom 3D scene



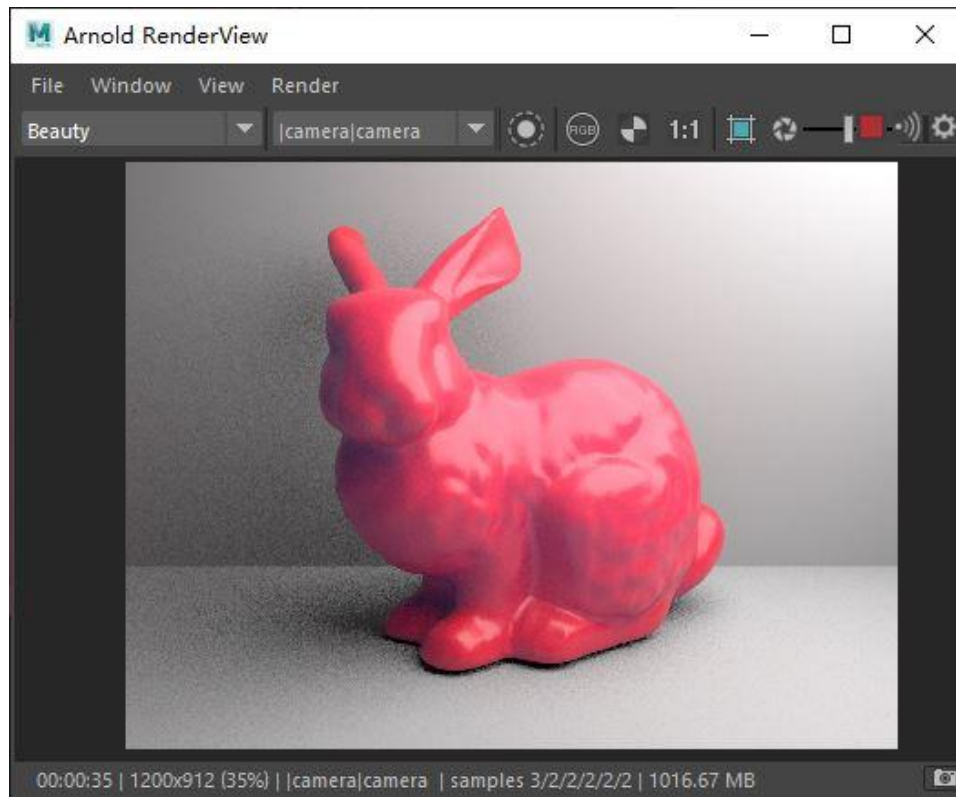
scene.mb

3. Open hypershade, and assign the “StylitRed” material to the models



StylitRed Material

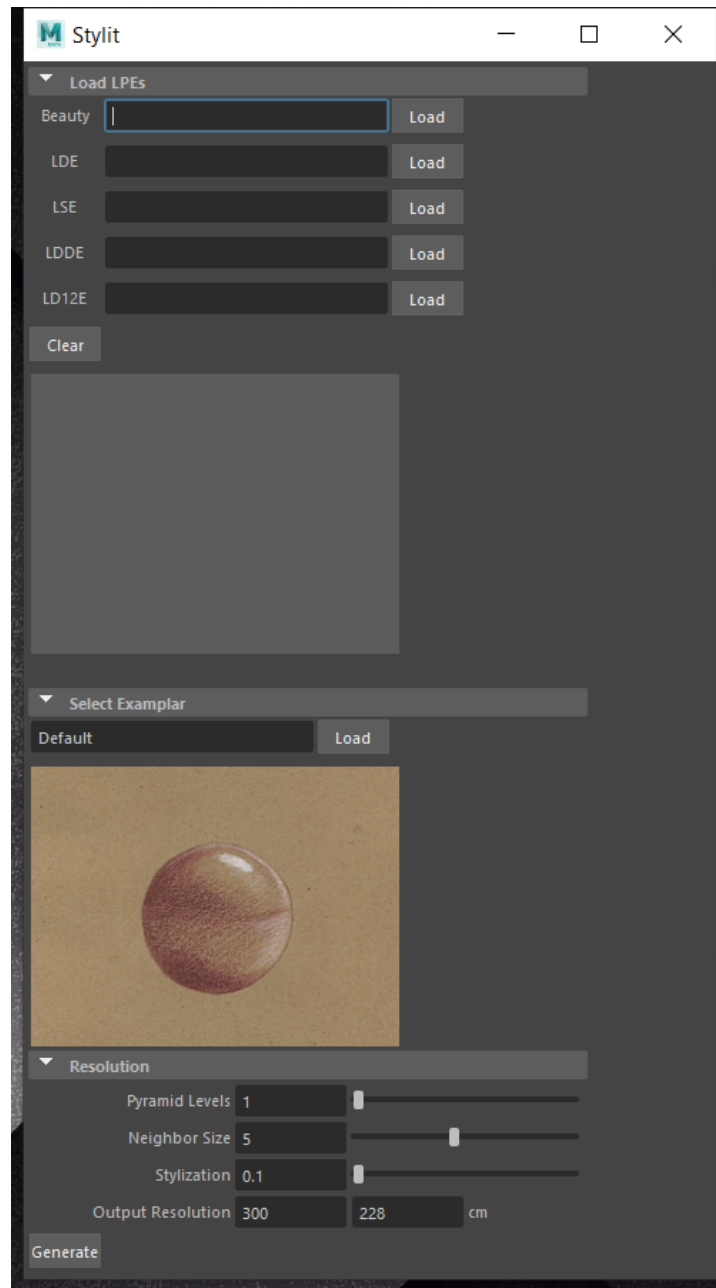
4. Open Arnold and you will see the below scene
5. Render and save the five LPE images in Arnold by changing the AOVs



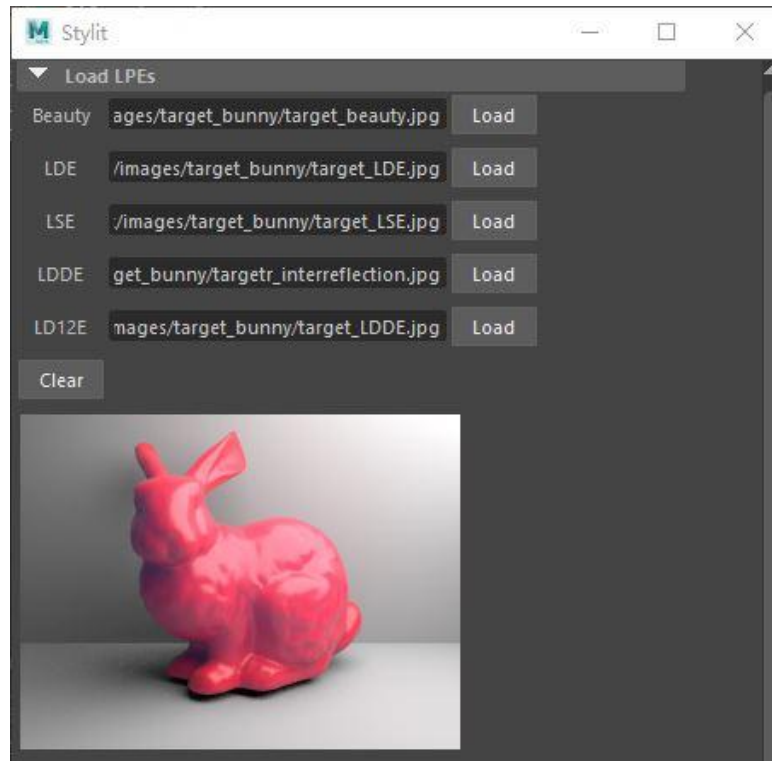
Arnold Renderer

## STEP2: Stylization

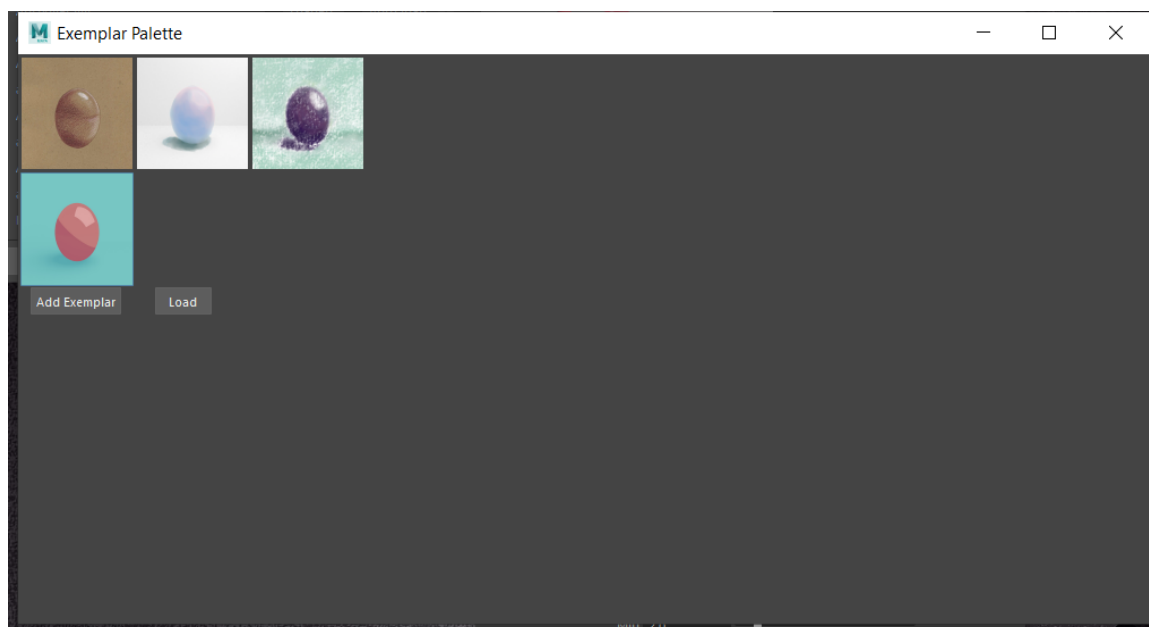
### 6. Open Stylit Plugin



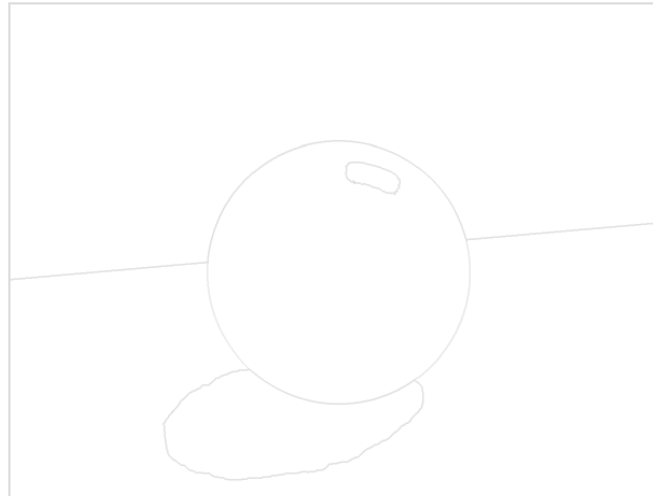
### 7. In 'Load LPEs' section, load the five LPEs you just created in STEP1



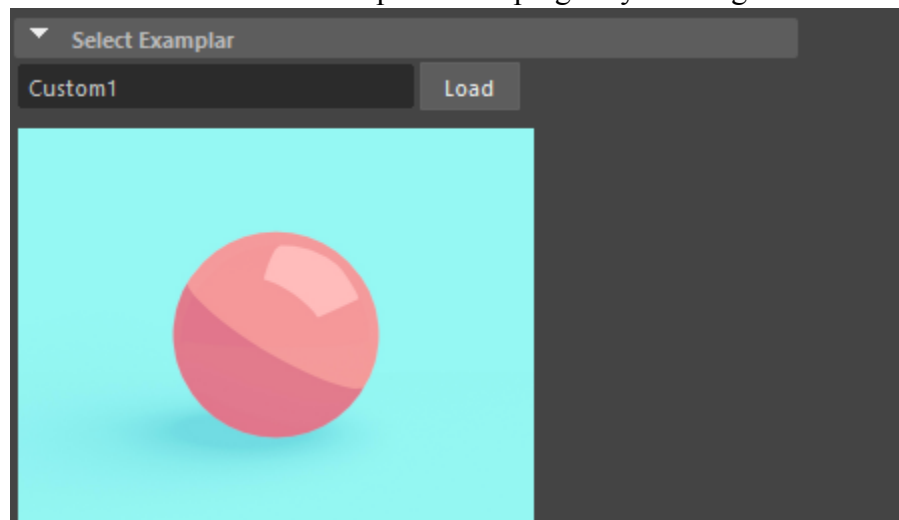
8. In 'select Exemplar' section, you have two options:



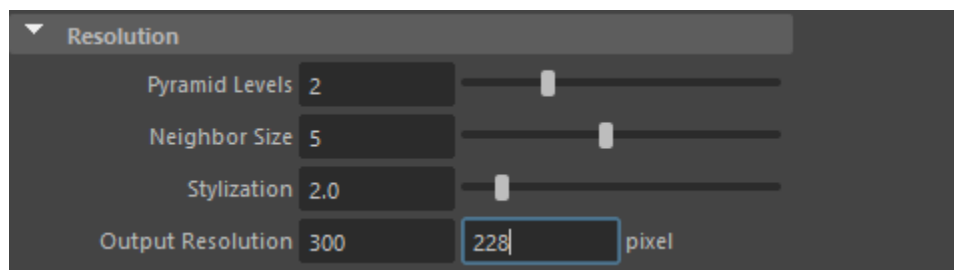
- select a default exemplar
- create your own exemplar using the below image we provided



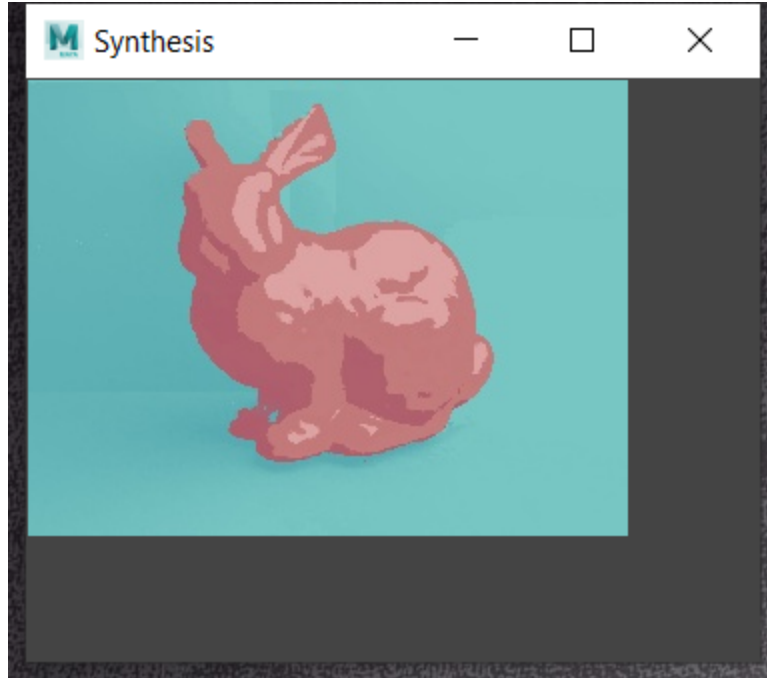
9. Load the default or custom exemplar to the plugin by clicking on 'Load'



10. In 'Resolution' section, modify the parameters



11. Finally, click *Generate*, it will show the result in a few minutes and also automatically save the output to the same directory as the beauty LPE image.



Result

## 6. Discussion

---

- **Accomplishments**

We developed an authoring tool that can reflect the designed visual effect to the model. The tool can provide an artistic visual effect for 3D models, including complex objects like landscape, with very low cost, especially suitable for independent developers. Users just need to render five LPE images and draw a very simple “sphere on the table” scene, then a synthesized artistic image with adjustable resolution and size can be generated.

Most features we planned have been implemented, except that we didn’t combine our plugin with Arnold. We discussed with Dr. Lane about that feature and felt it unnecessary to combine these two because we have already had a complete workflow for the artists. Instead, we polished our GUI, implemented acceleration NNF and ran tests in different scenarios(shown in appendix).

- **Design Changes**

Initially, we match the pixels in source and target image pixel by pixel. But it takes dozens of hours to generate an image with a megapixel. To accelerate the synthesis process, we implement multi thread, which can use up to 9 threads work synchronously. Furthermore, we improve the pixel by pixel retrieval of nearest neighbours to PatchMatch with integrated support for masking [Barnes et al. 2009]. So that our plugin can generate an image with a megapixel in a few minutes.

- **Total Man-Hours worked**

Li Zheng: 74hr

Keyi Yu: 87hr

- **Future Work**

1. We will add a feature to our plugin so that users can process a batch of models and examples at once with a script.
2. The synthesis can be further accelerated by using GPU, so it can be used interactively in real time.
3. Additionally, we may integrate it to Arnold and Iray to make it more convenient to use.
4. We will spawn a separate thread for the GUI which solves the problem of Maya stuck. In this way, the GUI can display a progress bar and intermediate stylized images. Users can kill the generating process anytime.

- **Third Party Software**

Arnold: Arnold is an advanced Monte Carlo ray tracing renderer built for the demands of feature-length animation and visual effects. We use it to generate LPE images.

OpenCV: Provides a real-time optimized Computer Vision library, tools, and hardware. We use it to process images.

Maya: We develop our plugin for Maya with MEL and C++.

## 7. LESSONS LEARNED

---

1. We write the *showWindow* command in C++, it can execute in Maya 2020, but doesn't work in Maya 2019. It could be a bug.
2. Since our authoring tool is for rendering, during the investigation, we find many renderers that can be integrated into Maya, like Iray, Vray, Arnold and Octane. They can be used for both realistic and unrealistic rendering.
3. OpenCV is very useful for image processing. We use it to read, write and resize images.
4. In OpenCV, *pyUp* and *pyDown* are designed for images pyramids. They are different from *resize* function in OpenCV.



## APPENDIX:

New tasks after meeting with Dr. Lane

1. UI (Li)
  - a. Display input image
  - b. Parameters:
    - i. pyramid level
  - c. 5 Load
2. Code (Keyi)
  - a. Resize source images according to the size of exemplar
  - b. UI relative code
3. Test
  - a. Whether the size of the sphere would affect the result (Keyi)
  - b. Landscape (Li)
  - c. Higher resolution vs time (Keyi)
  - d. Target size same with source size (Keyi)
  - e. unrealistic render validation (Li)
4. Demo
  - a. Generating LPEs
  - b. high resolution results
  - c. landscape results