INF584 - Image synthesis (2021-2022)
Final project - implementation of a research paper

StyLit [1]
Illumination-Guided Example-Based Stylization of 3D Renderings

Antonin Wattel

1 Introduction

The goal of this project is to work on the implementation of a research paper in the field of image synthesis. We choose the paper *Stylit* [1], which proposes an approach for exemple-based stylization of 3D renderings, based on light propagation in a 3D scene. Our implementation features a global illumination rendering algorithm, as well as a simple version of the example-based synthesis algorithm described in section ***3.3*** *Synthesis algorithm*, *Algorithm 1.* of the paper. The rendering algorithm is a Montecarlo pathtracer, based on source code of a raytracer provided during the INF584 course by Tamy Boubekeur.

The implementation (source code, resource files, required libraries, compilation instructions) is available here.

2 Paper overview

In this section, we provide an overview of the approach presented in the paper, referring mainly to section *3 Our Approach* of the paper.

The main idea of example-based stylization is the following. Given 3 multichannel images A, A', and B, the goal is to synthesise an image B' such that the style of A' is transferred to B', using the similarity between A and B, (Fig. 1)

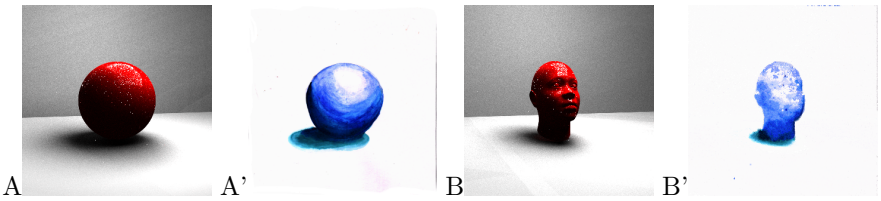
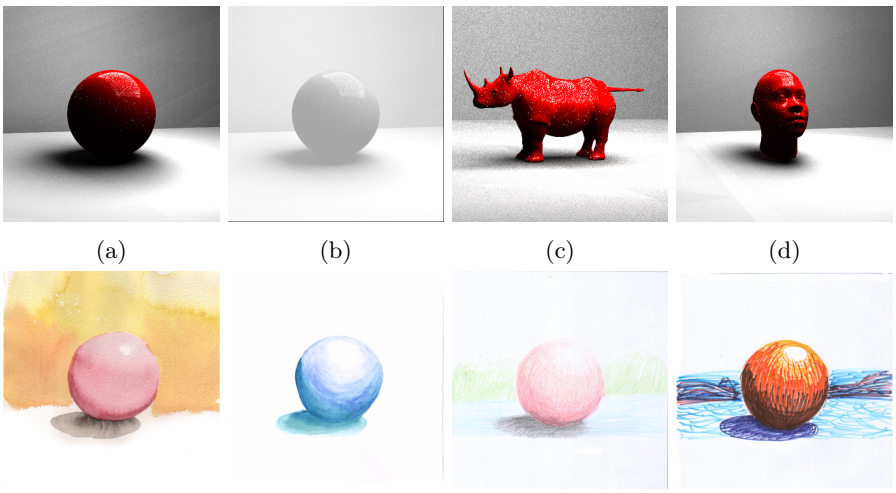


Figure 1: The concept of image analogies

2.1 Workflow

The approach requires two main components: A renderer, to compute light propagation in the scene, and the example based algorithm, taking images A, A' and B as input. First, a simple 3D scene, such as "a sphere on a table" is built and rendered using a global illumination algorithm (Fig. 1). Then, an artist paints (digitally or not) on top of a low contrast version of this rendered scene, keeping alignment (Fig. 2). Then, a target scene is rendered using the renderer (Fig. 2). Finally, we can execute the example-based algorithm.



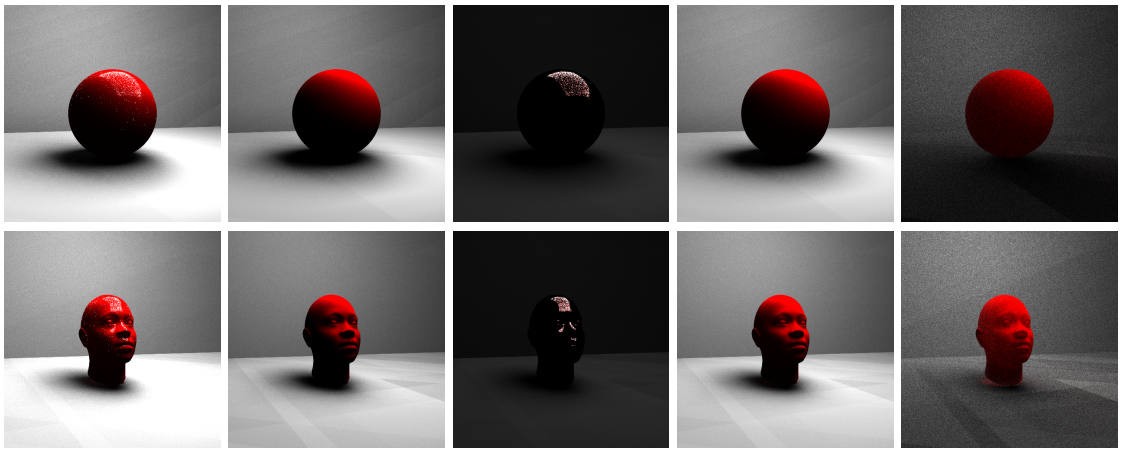
(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 2: **Workflow** (a) Global illumination render of "a sphere on a table" (b) Low contrast version of the rendering used by the artist (c)(d) Examples of complex target scenes (e)-(h) Examples of style exemplars

2.2 Light Path Expressions

An important aspect of the approach is to use illumination information of the 3D scenes to guide the synthesis algorithm, in order to reproduce the richness and illumination of artist-made style images, as opposed to previous approaches that used only color or normals. This is done through the use of Light Path Expressions (LPEs), separating various illumination effects into distinct buffers. Using a global illumination algorithm, the idea is to isolate different illumination effects, such as diffuse or specular lighting, direct and indirect lighting. (Fig.3).



(a) (b) (c) (d) (e)

Figure 3: **Light Path Expression images** (a) full global illumination render, (b) direct diffuse, (c) direct specular, (d) first two diffuse bounces, (e) diffuse interreflection

2.3 Algorithm

We define $\mathbf{A} = \{A, A'\}$, $\mathbf{B} = \{B, B'\}$ and a weight μ controling the influence of guidance. For any image I, I(p) denotes a feature vector at a pixel p, a possibly multichannel concatenation of all pixels in a square patch of width w centered at the pixel p.

A first algorithm, proposed by Hertmann et Al. in *Image analogies* [2] is the following. 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' to minimize the following energy.

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

Subsequent work [3] showed that better results, preserving high level structures can obtained by minimizing

$$\sum q \in \mathbf{B} \min_{p \in \mathbf{A}} E(\mathbf{A}, \mathbf{B}, p, q, \mu) \tag{2}$$

This is done through an EM-like iteration executed from coarse to fine resolution.

input : multi-channel images $\mathbf{A} = \{A, A'\}$ and $\mathbf{B}_k = \{B, B'_k\}$
output: synthesized target image B'_{k+1}

for *each pixel* $q \in \boldsymbol{B}_k$ **do**
  $NNF(q) = \underset{p \in \mathbf{A}}{\operatorname{argmin}} E(\mathbf{A}, \mathbf{B}_k, p, q, \mu)$

for *each pixel* $q \in \boldsymbol{B}_k$ **do**
  $B'_{k+1}(q) = Average(\mathbf{A}, NNF, q)$

Algorithm 1: EM-like iteration used to minimize energy

Where B'_k denotes the current pixel values stored in B' and $B'_k + 1$ 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.

This technique, however, can lead to a wash-out effect. To mitigate this problem, the authors of the paper propose an idea based on reverse NNF retrieval, consisting in retreiving a best matching target patch for each source patch. This has the advantage of enforcing uniform usage of source patches. In order to avoid matching sources patches to inapropriate locations on the target image, a feasible error budget T is computed. We can fit the sorted error values of sources patches assignments to a hyperbolic curve and compute a knee point k, past which assignments are considered erroneous. This technique has the advantage of maximizing the number of used source patches while enforcing the error budget constraint. Then, the final algorithm is an EM-like iteration similar to Algorithm 1, where the NNF creation is replaced. The algorithm can stop when all target patches are covered. Moreover, Patchmatch [4] can be used to speed up the retrieval.

3 Implementation

3.1 Renderer

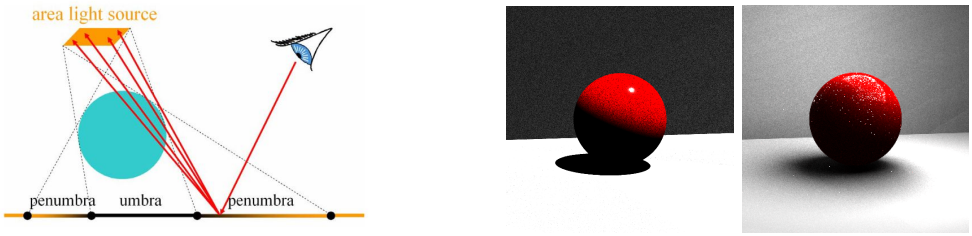
Before implementing the example-based stylization algorithm, it is necessary to have a global illumination algorithm. We make use of the implementation of the raytracer provided in the INF584 course to build a simple Montecarlo path tracer.

Area lights and soft shadows

First, we implement area lights support for our ray tracer, in order to render soft shadows. For a simple directional light, shadows are created by using a visibility term. By throwing a ray from the first ray-intersection in the light direction, we can check if it intersects an object in the scene, in which case we have a hard shadow. For our area light, we use the same principle, but instead, we throw a ray from the intersection to a random position on the area light. We repeat this n times for each intersection and divide by n, to get a montecarlo approximation of the shadow (Fig. 4). Noise reduces when increasing n. To uniformly sample a random point on the area light, which is a triangle mesh, we first sample a random triangle abc on the mesh, with a distribution weighted by the areas of the triangles. We can then uniformly sample a point p on this triangle by sampling two random uniform numbers $r1$ and $r2$ between 0 and 1, and evaluating $p = (1 - \sqrt{r_1}a + \sqrt{r_1}(1 - r_2)b + \sqrt{r_1 r_2}c$, as seen in [6], section *4.2.* In our implementation, we use a simple rectangle made of two equal area triangles, and did not implement a distribution function using the triangle areas.

Monte Carlo Path tracer

In order to support antialiasing and indirect lighting, we extend our ray tracer to a path tracer. We draw inspiration from the following implementation [7].



(a) (b) (c)

Figure 4: (a)[8], Montecarlo soft shadows, (b) scene with one directional light, with hard shadows, (c) scene with one rectamgular area light, with soft shadows

Thanks to a BVH acceleration structure, it is computationally feasible to implement a decent simple path tracing algorithm, as seen in the INF584 course.

for each pixel in the image **do**
  pixel = black
  for a set of N randomly distributed locations D in the pixel **do**
    pixel $+= 1/N$ * path traced response though D

The path traced response is computed by recursion, at the ray intersection Y and modulated with the GGX BRDF at X, and divided by random sample PDF. To compute the random ray, we sample a random point on the hemisphere oriented about the normal N of the intersection point Y. The result is summed to the response computed from direct lighting modulated by the BRDF at X. We keep the branching factor to one and only sample one location of our area light, to avoid a too heavy rendering time. We use simple russian roulette to speed up the rendering.

In order to improve our rendering algorithm, there are several steps to take. Our random sampling being slow to converge, a next step would be to implement importance sampling for the microfacet GGX BRDF, using the approach described in [5]. Moreover, a denoising algorithm should be applied to the rendered image.

3.2 Light Path Expressions

The Image class is extended to support multiple channels of RGB images, to store the different LPE buffers. During the path tracing, we can fill these buffers by separating diffuse and specular light from the microfacet GGX BRDF, and taking into account the bounce we are currently computing. We obtain LPE buffers seen in Fig. 3. The computation of LPE channels does not impact the rendering time, as they are all being computed in the same time.

3.3 Example-Based stylization algorithm

Our implementation focuses on Algorithm 1. The inverse NNF retrieval and error budget estimation presented in the Stylit paper were not implemented.

First, in order to run several steps of the EM-like iteration algorithm from coarse to fine resolution, we use image pyramids (5). An next improvement will consist in applying a gaussian filter before downsampling the image. We compute image pyramids for the source and target images A, B, as well as style examplar A'. We perform n iterations of the algorithm for each pyramid level, from coarse to fine resolution and at the end we upsample the synthesised image B' to use it in the next finer iteration. To compute the NNF of a patch of pixels centered at pixel p, we use a brute force approach. A next improvement would be to use a more efficient search method, or as seen in the paper, to use the PatchMatch algorithm [4].

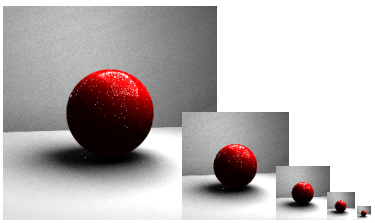


Figure 5: **Image pyramid**, 5 levels, downsampling by 2 (no gaussian filter)

4 Results

We show results for images of size $320 * 320$ pixels. Global illumination images are rendered with 200 samples per pixel, and 3 recursive bounces. Each multichannel image contains 5 LPE channels (Fig. 3). The stylization algorithm is run on 6 pyramid levels, with a downsampling of 2. For each level, we perform 4 iterations, with a patch size of 2 and $\mu = 2$. The algorithm is run CPU, with OpenMP parallelization.

4.1 Visual results

Figure 6: Example results

4.2 Time performance

Path tracer

The path tracer uses a BVH acceleration structure, as well as simple russian roulette with threshold 0.5. This allows to have renders in resonable time for the $320 * 320$ images we are working with . An implementation of importance sampling, and a denoiser would however be of great help to speed up and increase the quality of our renders.

Stylization algorithm

Using a brute force approach, our algorithm performs poorly, and it becomes impossible to work with images of a decent size. The next step would be to use a better search approach for the NNF retrieval.

model / pyramid level	5 (10*10)	4 (20*20)	3 (40*40)	2 (80*80)	1 (160*160)	0 (320*320)
face	2ms	13ms	200ms	3s	50s	13m

Table 1: Average Time performance for one EM-like iteration, using style exemplar 1 (pink and orange). These results are similar for other target images and style exemplars.

5 Conclusion

To conclude, we worked on the implementation of *Algorithm 1* seen in the *Stylit* paper. Our work consisted in building a montecarlo path tracer, in order to create multiple channel images, containing Light Path Expressions (LPE), describing different illumination effects of scenes. After rendering a simple scene, "a sphere one a table", painting a style exemplar for it, and rendering another more complex target scene, we can apply the example-based stylization algorithm to transfer the style of the exemplar to the target, while preserving important illumination features.

References

[1] akub Fišer and Ondřej Jamriška and Michal Lukáč and Eli Shechtman and Paul Asente and Jingwan Lu and Daniel Sýkora, "StyLit: Illumination-Guided Example-Based Stylization of 3D Renderings", "ACM Transactions on Graphics", 2016

[2] Aaron Hertzmann and Charles E. Jacobs and Nuria Oliver and Brian Curless and David H. Salesin, "Image analogies", SIGGRAPH Conference Proceedings,327–340, 2001

[3] Fišer, J., Lukáč, M., Jamriška, O., Čadík, M., Gingold, Y., Asente, P. and Sýkora, D. (2014), Color Me Noisy: Example-based Rendering of Hand-colored Animations with Temporal Noise Control. Computer Graphics Forum, 33: 1-10. https://doi.org/10.1111/cgf.12407

[4] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman. "PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing." ACM Transactions on Graphics (Proc. SIGGRAPH) 28(3), August 2009.

[5] Eric Heitz. A Simpler and Exact Sampling Routine for the GGX Distribution of Visible Normals. [Research Report] Unity Technologies. 2017. hal-01509746

[6] Osada, R., Funkhouser, T., Chazelle, B., Dobkin, D. (2002). Shape distributions. ACM Transactions on Graphics, 21(4), 807-832.

[7] Global Illumination and Path Tracing: a Practical Implementation, https://www.scratchapixel.com/lessons/3d-basic-rendering/global-illumination-path-tracing/global-illumination-path-tracing-practical-implementation

[8] Soft shadows image - MIT 6.837 Lecture notes, Monte-Carlo Ray Tracing, https://dspace.mit.edu/bitstream/handle/1721.1/86191/6-837-fall-2003/contents/lecture-notes/19_montecarlo.pdf