

Master Thesis



Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

Example-based Non-photorealistic Rendering using Game Engine

Jiří Burýšek

Supervisor: prof. Ing. Daniel Sýkora, Ph.D.
Field of study: Open Informatics
Subfield: Computer Graphics and Interaction
May 2019

Acknowledgements

Děkuji své rodině a přátelům za morální oporu, taktéž Honzům za jejich postřehy, Petce za jeho počítač a vedoucímu Danovi za pomoc a nekonečnou trpělivost.

Declaration

I declare I have accomplished my final thesis by myself and I have named all the sources used in accordance with the Guideline on ethical preparation of university final theses.

Prague, 24. May 2019

Abstract

This work presents a solution for video game scene art stylization in video game engine Unity. The example-based stylization implements the StyleBlit algorithm and is able to provide results for an arbitrary art style. A plugin providing simple integration of StyleBlit method is described and evaluated.

Keywords: Example-based stylization, Unity, Non-photorealistic rendering, Real-time stylization

Supervisor: prof. Ing. Daniel Sýkora, Ph.D.

Abstrakt

Práce prezentuje praktické řešení výtvarné stylizace videoherní scény v herním enginu Unity. Metoda stylizace užívá výtvarné předlohy a je založena na algoritmu StyleBlit, přičemž je schopna napodobit libovolný výtvarný styl. Výsledný plugin, jenž zajišťuje uživatelsky přívětivou integraci metody StyleBlit, je v této práci popsán a zhodnocen.

Klíčová slova: Stylizace dle předlohy, Unity, Nefotorealistické zobrazování, Stylizace v reálném čase

Překlad názvu: Nefotorealistické zobrazování v herním enginu s využitím výtvarné předlohy

Contents

1 Introduction	1	3.3 Example-based stylization	35
2 Motivation	3	3.3.1 StyLit	35
2.1 Photorealistic and non-photorealistic rendering	3	3.3.2 Expressive Animation of 2D Rigid Bodies	36
2.2 Photorealism and non-photorealism in video games	5	3.3.3 ToonSynth	37
2.2.1 Cel-Shading	6	3.3.4 Facial animation	38
2.2.2 Discussion on non-realistic graphics variability in video games	7	3.3.5 Stylizing video	39
2.3 Overview of non-realistic video game graphics in history	11	4 StyleBlit	41
2.3.1 Games utilizing own style	12	4.1 Overview	41
2.3.2 Games imitating established art style	20	4.2 Algorithm	42
2.3.3 Games imitating realistic craft	25	4.3 Implementation	44
2.3.4 Goals	28	4.4 Unity	47
3 Related works	31	4.5 Results and evaluation	50
3.1 Specific art style imitation	31	4.5.1 Performance	51
3.2 Stylization based on contained patterns	33	4.5.2 Comparison with vanilla version	53
		4.5.3 Practical evaluation	56
		4.5.4 Limitations and future work	61

5 Conclusion	63
A Bibliography	65
B Project Specification	69

Figures

2.1 The Adventures of Tintin	3	2.17 Killer Is Dead	16
2.2 Spider-Man: Into the Spider-Verse	4	2.18 The Witness	17
2.3 Uncharted 4	5	2.19 Lara Croft GO	18
2.4 Jet Set Radio	6	2.20 Life Is Strange	19
2.5 Cel-Shading	6	2.21 The Walking Dead	19
2.6 The Simpsons Game	7	2.22 Ōkami	20
2.7 Pokémon Sword and Shield	8	2.23 Ink wash painting	21
2.8 Borderlands 3	9	2.24 Assassin's Creed Chronicles: China	22
2.9 Prince of Persia	10	2.25 Dreams	22
2.10 Super Mario Galaxy 2	11	2.26 Valkyria Chronicles Remastered	23
2.11 Team Fortress 2	12	2.27 Ni no Kuni II: Revenant Kingdom	24
2.12 Half Lambert illumination	13	2.28 XIII	24
2.13 Journey	14	2.29 11-11: Memories Retold	25
2.14 Love	14	2.30 Kirby and the Rainbow Curse .	26
2.15 Inside	15	2.31 Yoshi's Woolly World	27
2.16 Volumetric lighting	16	2.32 Paper Mario: Color Splash	28
		2.33 League of Legends supplementary visuals	29

2.34	Rebelle painting	30	4.9	Performance test sample	51
3.1	Watercolor painting imitation . . .	32	4.10	Plugin vs. vanilla comparison . .	53
3.2	Calligraphy imitation	34	4.10	Plugin vs. vanilla comparison . .	54
3.3	Chinese paintings imitation	34	4.10	Plugin vs. vanilla comparison . .	55
3.4	StyLit	36	4.11	Watercolor scene exemplar	56
3.5	Animated sequence imitation . . .	36	4.12	FIT game exemplar	57
3.6	ToonSynth	37	4.13	Initial multi-material exemplar	58
3.7	Facial animation stylization	38	4.14	Watercolor exemplar	59
3.8	Video sequence stylization	39	4.15	Pencil drawing	60
4.1	StyleBlit overview	42	4.16	Flat surface limitation	61
4.2	StyleBlit method	43			
4.3	Seed hierarchy	44			
4.4	Guidance in Unity	47			
4.5	Chunk distribution	48			
4.6	Initial result	49			
4.7	Unity multipass degradation . . .	50			
4.8	First plugin exemplars	50			

Tables

4.1 Performance test on CPU Intel Core i5-6300HQ 2.30 GHz with GPU Intel HD Graphics 530.	52
4.2 Performance test on CPU Intel Core i5-6300HQ 2.30 GHz with GPU 2GB GeForce GTX 950M.	52
4.3 Performance test on CPU Intel Core i5-7600K 3.80 GHz with GPU 6GB GeForce GTX 1060.	52



Chapter 1

Introduction

Non-photorealistic graphics gains higher prominence in a video game industry each passing year. Despite this, the overall variability of stylized video games is limited, both technically and conceptually. The goal of this work is to provide a solution for an arbitrary stylization of a game scene in the Unity game engine environment, usable for virtually any video game project. This work provides an introduction into non-photorealistic stylization in video games and list a comprehensive catalog of stylized video game projects. This overview aids in stating the issues of stylized video games and goals for future video games, formulating the motivation of this work. This leads to the development of the practical solution, which implements the StyleBlit algorithm able to provide arbitrary example-based stylization that is fit for a limited computational budget. The implementation is tested and evaluated for its deployment in video game development.

Chapter 2

Motivation

2.1 Photorealistic and non-photorealistic rendering



Figure 2.1: Tintin feature movie used motion capture and photorealistic rendering to blend comics characters into a realistic world. Source: *The Adventures of Tintin*, 2011

3D computer graphics may represent scenes in two established divisions: Either the output image is ideally indistinguishable from captures of real-life scenes, or the representation is stylized, possibly as an existing art form. Despite being valid in computer graphics sphere only, this branching is easily

recognizable and generated imagery may be categorized in this manner at first glance.



Figure 2.2: Spider-Man: Into the Spider-Verse movie used non-photorealistic rendering and found its style in comics drawings. Source: *Spider-Man: Into the Spider-Verse*, 2018

Obviously, non-photorealistic imagery is a very broad term, however further categorization is not an easy task and one without solid foundations, perhaps for the fact, the 3D non-photorealistic imagery has relatively short tradition, or simply because the aim of photorealistic imagery is so specific itself, remaining myriads of graphical styles defy other unions than being its mere opposites, making it hard to find footing for divisions. This becomes apparent in further sections, where many non-realistic styles are discussed. Non-realistic designs reach from imitating traditional art styles like painterly, to introducing their own styles without any label. As is to be seen in various instances of non-photorealistic styles in video games, the term non-photorealistic rendering is problematic, since various solutions achieve the non-realistic image by other means than rendering techniques, be it models, textures or filters. For the sake of avoiding confusion with techniques used in listed cases, the following chapter will refer to using terms non-realistic graphics or design, instead of established non-photorealistic rendering.



Figure 2.3: Uncharted 4 is a primal example of photorealistic rendering in video games, while being arguably one of the best looking games of its generation. Source: *Uncharted 4*, 2016

2.2 Photorealism and non-photorealism in video games

One of the primary applications of computer graphics involves the video game industry and the division of generated imagery stays very much the same: The graphical design of 3D video games generally takes either of two directions: It strives for photorealism or imagines very much different stylization. As computer graphics, as a field of study, and technical potential evolves, this division comes somewhat late. That is no surprise, for the first years of video games set in a 3D environment the technical possibilities of graphical interpretation were so limited, the stylization was majorly based upon object modeling, hence pinpointing the turning point for non-realistic graphics in video games is dubious.

Similar to the race for photorealism, non-realistic graphics is limited by the technical status quo as well. For that matter, this imagery in video games became quickly a problem of cel-shading. Introduction of cel-shading technique to video games came with the game *Jet Set Radio* (see fig. 2.4) in 2000 [Luq12].

Since then, it has been used to imitate a wide range of authentic visual styles. For one, it was used as a comic visual style, e.g. *XIII*, similarly a cartoon animation like in *The Simpsons Game* (figure 2.6) or even ink



Figure 2.4: Jet Set Radio videogame, released in 2000, is a primal example of cel-shading technique. Source [Luq12]

painting sumi-e in *Ōkami* videogame.

2.2.1 Cel-Shading



Figure 2.5: A teapot model rendered photorealistically (left) and using cartoon-looking method (right), which later became known as cel-shading. Source [Dec96]

This technique's goal is to visualize a model in a way that is common in cartoons and other visual media that effectively reduce the complexity of any given object they capture. The technique simplifies the illumination model, leaving input object with flat colors, discarding color gradients and instead presenting layers of singular colors, while any contour or distinctive

color transition is enhanced with a thick line. Naturally, this technique is prone to different approaches to its implementation and enhancement. One representative is a video game series *Borderlands* (figure 2.8), which makes use of cel-shading while supporting it with some level of a peculiar textural and model stylization. Other games, for example, *Assassin's Creed Chronicles* series, blends this technique with a variety of other different rendering techniques and particle effects, creating its own design with carved-like characterization.



Figure 2.6: The Simpsons Game utilized the cel-shading technique to cope with original drawings being partly invariant to different perspectives (e.g. hair of the characters). Source: *The Simpsons Game, 2007*

Although there are many different approaches towards cel-shaded style, the core design remains still the same and being one of the most common non-realistic graphics technique in video games, effectively limiting the overall variety of stylization in video games.

2.2.2 Discussion on non-realistic graphics variability in video games

Even though the technological limits are ever-pushed back, there seem to be limits of imagination in regard to non-realistic game graphics. As per any graphical concept, there is an immense difference of realization possibilities in terms of deployment: Using video game terminology, comparing pre-rendered graphics and graphics rendered in real-time, the feasibility constraints reach entirely different levels. Put in plain perspective, for each technique used in the computer graphics field, the game developer figuratively asks a question, whether this technique's process might do in 60 frames per second, therefore video game graphics often uses different rendering mechanics, frequently

relying on heuristics and solutions only imitating the exact or expected results. One example is reflection visuals, using methods such as cube mapping or screen space reflections to create an unfaithful, yet feasible reflection without providing an exact one.



Figure 2.7: Pokémon Sword and Shield installments' may be perceived as an evolution to the cel-shading style, adding finesse to the formula. Source *Pokémon Sword and Shield*, 2019

It is not without interest that big budget video games, or AAA games, as they are called, usually race for ideal photorealistic rendering. In reality, the number of top-budget games released each year offers just a small fraction of games using non-realistic graphics. This is apparent from data of top budget games [Hod19].

One primal example of AAA game to utilize stylized graphics might be the beforementioned *Borderlands* series (figure 2.8). The biggest games delivering in this objective, non-realistic design, often come from Japanese company Nintendo, which houses brands like *Mario* and *Pokémon*. Those series offer some other examples for the given topic, naming *Super Mario Galaxy*, that is portrayed in generic 3D cartoon style, or *Pokémon*, both *Sword* and *Shield* editions (see figure 2.7). This *Pokémon* installment delves into cel-shading technique once more, but reformulates the idea: It does flatten the color surfaces of the model, only this time the areas mix a limited palette of colors, further enhancing areas with various gradient intensity. This approach often yields great results as it shows human-like touch to the style; the system is simple enough to imitate cartoon animation and at the same time complex enough to mask the 3D nature of the models.

These are some examples to prove the video game industry evidently shows



Figure 2.8: Fair share of Borderlands series' fame comes from its non-realistic art style. Source *Borderlands 3*, 2019

endeavors to capture various visual styles and yet there is a lack of titles that proved in their respective fields. As a customer, a player, one would expect common traditional art styles, for instance, watercolor painting, to break into the design of many video games, nevertheless to find a video game that successfully captures this art style is next to impossible. As discussed, the obvious drag is technological advancement. Additionally, it was already pointed out that big budget titles usually pursue photorealism, which, in a collision with technological progress in certain fields, might prove fatal. To simplify the matter, it can hardly be expected for non-realistic graphics in video games to advance in any measure close to realistic rendering, because there is neither enough resources nor demand for this kind of visual design. Non-realistic graphics is commonly reserved for indie games or at least side projects of major companies. To further elevate the speculation, even if non-realistic graphics becomes a topic for a big budget project backed by great working force, it is still no guarantee of any progress. Looking back at the reboot of *Prince of Persia* from 2008 (figure 2.9), the game itself came from hugely successful series, traditional name and one of the biggest companies in the video game industry. Yet the art style still benefits from the same old cel-shading style with somewhat realistic textures. As video games grow large budget-wise, the less experimenting they usually take. This is understandable, as they aim at the largest cut of a potential audience, including so-called casual gamers. This phenomenon is well known in the gameplay of such games, the same as their story or narrative. The notion behind this may very well be to supply a player with just enough novelty as he might possibly take, which, for the broad audience isn't much. Nowadays it seems genuinely unlikely to see a major title with watercolor-imitating graphics, as any such attempt would compromise the accessibility of the title



Figure 2.9: Prince of Persia represents one of the big budget games featuring non-realistic style: Another spin off the cel-shading style. Source *Prince of Persia*, 2008

for the masses. From the perspective of a developer or publisher respectively, it makes sense to “play it safe” with a release of something familiar, which may ironically be another cel-shaded title. This attitude effectively limits the mass audience’s familiarity with new attempts in non-realistic design, thus completing the catch-22.

Another explanation for the lack of stylized video game titles is the technological and artistic duality of such. Analogically to other industries, the central figures engaged in the game development may be uneducated or simply unaware of possibilities in certain related fields comprising the game development as a whole. Head designer may lack technological background altogether, while still being in charge of decisions in the game development art-wise. Naturally, the problem deepens with the scale of any such project. To back this matter with an example, the game segments that delve into experimental parts, say during hallucinations or dreams of the protagonist, seem all very similar throughout many games of certain era, be it *Far Cry*, *Arkham* series, *Metro* or *Mass Effect*: Whenever those games visually break the graphical style in those segments, they seem to remix colors, imitate malfunctioning TV screen, realign geometry, or use plain filters as if that was something the art designer can approach from his perspective. The flickering, color shift, jittering, it is all there, even though from the perspective of graphics engineer, this seems as thinking inside a box. All in all, this vacuum in the variety of stylization might very well be a symptom that is in its general sense common across different industries and simply comes from the unfamiliarity with given field despite being a figure responsible for it.



Figure 2.10: Super Mario Galaxy 2 follows the graphical stylization of dated kids-friendly platform video games, its style being more of a generic one. Source *Super Mario Galaxy 2*, 2010

2.3 Overview of non-realistic video game graphics in history

This section provides an extensive overview of many video games in relation to their unique graphical style. As repeatedly mentioned, the prominent technique remains to be cel-shading from [Dec96]. Generic approach to cel-shading is to render object's outlines and edges as uniform ink lines while flattening surfaces where shades are presented as layered uniform color areas. Even though the steps are subjects to change, the overall image is instantly recognizable, therefore the following overview lists mostly those video games to strive for unique image or art style imitation.

The overview is divided into two sections: Firstly, the games built around their particular style, which is unlike any existing visual style or technique from real life. Secondly, there are video games to try and capture a visual style that has some level of tradition in real life. Those two categories may seem to overlap, which is usually caused by selected techniques, that are very much similar to each other.

2.3.1 Games utilizing own style

Team Fortress 2



Figure 2.11: Team Fortress 2 pioneered the comic style reminding of plastique figures. Source *Team Fortress 2*, 2007

Team Fortress 2 uses a visual style to remind of the early to mid-20th-century commercial illustrators J. C. Leyendecker, Dean Cornwell, and Norman Rockwell. The resulting characters may be described as plastic figures of sorts. This is achieved by combining a number of techniques [MFE07]. Thanks to the openness in regards to the development of the game, it is possible to study used techniques in detail.

Notable technical inspiration for the developed style is the Gooch shading as described in [GGSC98]. The idea behind this technique is to render surfaces in a shift from “warm” to “cool” hue. Intuitively, the warm color tints the surface facing the light source and vice versa for the cool color. The result is traditionally outlined in black ink as for cel-shading. This technique was originally developed for technical illustration, though eventually found its way into a number of video games.

Valve, developer of Team Fortress 2, is also renowned for popularizing Half Lambert illumination (see fig. 2.12), which is worth mentioning in relation to other games in the list.

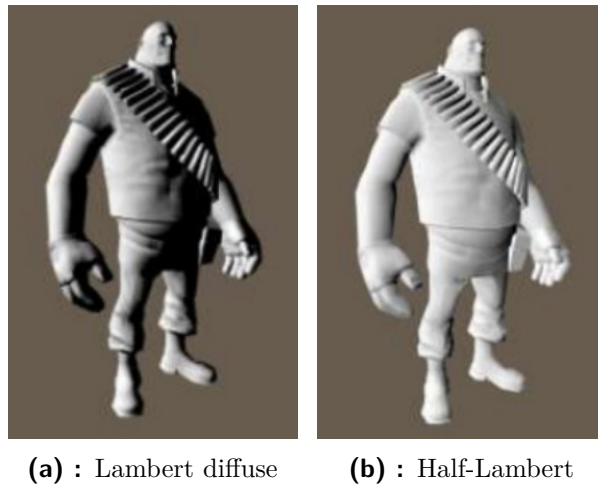


Figure 2.12: Visualisation of Lambert and Half-Lambert (used in Team Fortress 2) diffuse illumination. Source: *sfdm.scad.edu*

$$k_d \left[a(\hat{n}) + \sum_{i=1}^L c_i w((\alpha(\hat{n} \cdot \hat{l}_i) + \beta)^\gamma) \right] \quad (2.1)$$

where L is the number of lights, i is the light index, c_i is the color of light i , k_d is the albedo of the object sampled from a texture map, $\hat{n} \cdot \hat{l}_i$ is a traditional unclamped Lambertian term with respect to light i , the scalar constants α , β and γ are a scale, bias and exponent applied to the Lambertian term, $a()$ is a function which evaluates a directional ambient term as a function of the per-pixel normal \hat{n} and $w()$ is a warping function which maps a scalar in the range of 0..1 to an RGB color.

Equation 2.1 is view-independent lighting term used in Team Fortress 2. Lambert reflectance is used as a model for diffuse reflection. Constants of scale, bias and exponentiation applied to the Lambertian term cause the lighting to fade with gentle graduality on the object model, preserving the shape on the rear surface as apparent on figure 2.12.

■ Journey

Uniquity of Journey roots from the utter simplicity of the game world, where environment and game objects are often depicted as mere silhouettes. Shadows cast inside the game world often cover huge areas without complicating the density of the scene. Yet the graphic design doesn't seem underwhelming, as it makes great use of soft shading, ease color gradients and lighting, all in combination with widespread particle effects. The absence of some traditional



Figure 2.13: Journey displays minimalistic, yet very original art direction.
Source: *Journey*, 2012

artwork model used in the design of Journey accents the achievement of the game's style: The still images might seem handcrafted without easy saying of what makes the style tick.

■ Love



Figure 2.14: Despite not being an aim of the stylization, Love's art design comes close to watercolor painting style. Source: *Love*, 2010

Small game Love's art design uses a rather simple approach as each frame is first drawn into an off-screen buffer and, subsequently, the frame undergoes effect filtering and color correction [Ste]. Resulting image materializes fuzzy

or wobbly contours and shifting color tones. The outcome reminds of non-complex imitation of watercolor painting while maintaining enough of the original style.

■ Inside

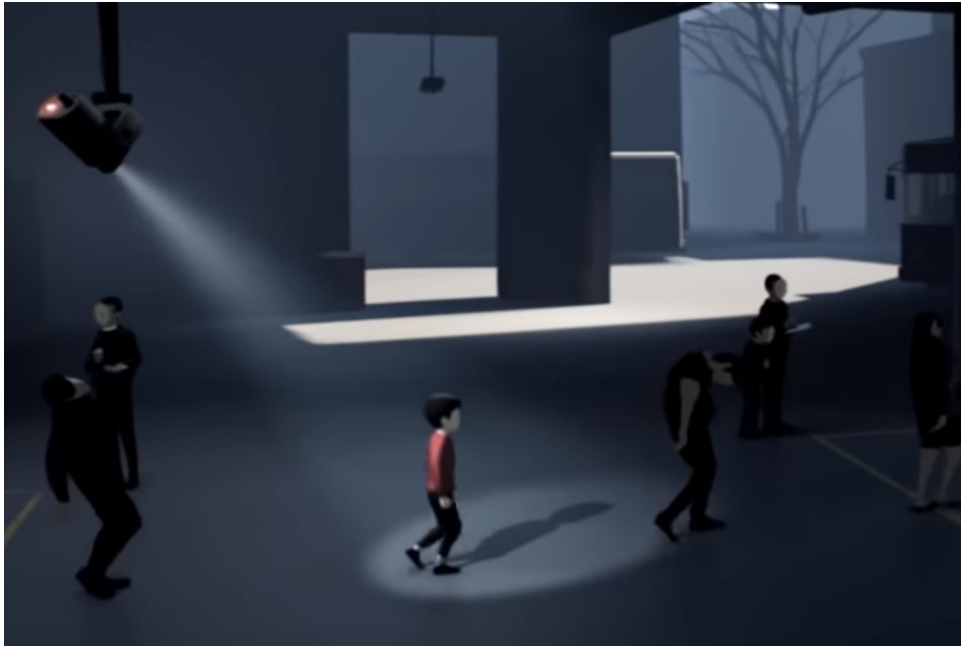


Figure 2.15: Inside shares graphical similarities with Team Fortress 2, although the mood and settings are entirely different. Source: *Inside*, 2016

Inside focuses on subtle visuals with emphasis on silhouettes, thus its environment draws a comparison to *Journey* mentioned earlier. There are many techniques the game utilizes [GDC16], from extensive use of bloom effect, smooth color grading to volumetric lighting. The game heavily relies on fog effects, which allow the light to create distinguishable volumetric shapes, including eclipsing the light source locally to diminish the illumination intensity in any such area (see figure 2.16). The shading varies for different sorts of objects and situations. Scenes make use of Bounce Lighting, which is similar to the lighting model of Team Fortress 2, Half Lambert, discussed earlier, and as may be observed, the similarity makes for the games being very close in terms of the overall style. The Bounce Light reduces the steep fall off of object illumination, thus a point light source of this manner recalls more of an area light. The plastique style of the whole image is contributed by a number of ambient occlusion decals, as they differ for various object nature. For instance, boxes handle ambient occlusion in a particular way, accenting the box's sides. Although subtle, there is a number of levels in the game that lean on reflections, usually accommodated by screen space

reflections. Their usage is somewhat special in the matter of possibilities and simplification as Inside is a 2.5 game, which makes for those solutions, that may be naturally different than those of 3D games.

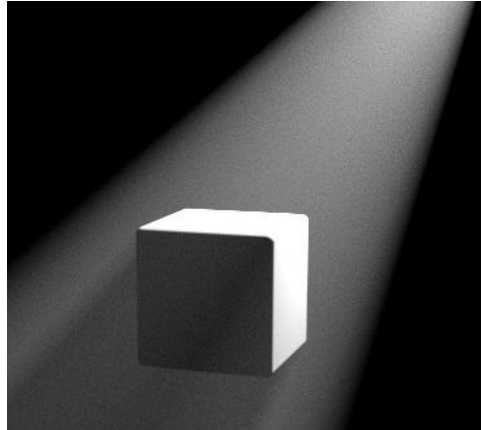


Figure 2.16: Inside relies heavily on volumetric lighting and fog effects. Source: *segoinsulation.com*

■ Killer Is Dead



Figure 2.17: Killer Is Dead is obviously a Japanese game. Source: *Killer Is Dead, 2013*

Killer Is Dead's graphical design may be perceived as an evolution to the cel-shading style. It is far more sophisticated, outlines take on different contrast colors and the style doesn't suppress any reflections, on the contrary, it enhances them in the overall image in a form of specular highlights. Flat colored surfaces are uniform once again, however, they are far more layered

and delicate. All in all, it is safe to say this take on the toon style is very much unrivaled.

■ The Witness



Figure 2.18: A "stylized realism" of The Witness. Source: *the-witness.net*

The Witness makes for an argument in an issue stated earlier: The rendering does not necessarily shape the stylization of the game. To be precise, The Witness' art design is, in developer words "stylized realism" [And14], without resorting to unorthodox technicalities on the rendering side. For one, the stylization of the game world comes from deliberately reducing the complexity of environment and game objects and adding rough cuts to the geometry. In certain cases, the result basically comes close to any respective object model from a dated 3D video game. This design choice coincides with the rendering technique, that plays on the fact that the game itself is very static. Excluding grass and foliage, there are relatively very few movable objects in general. This fact comes along with the choice of attempted global illumination [Blo10]: The system in place makes use of precomputed lightmaps to illuminate the environment, creating mellow and subtle visuals that feel very tender in the gameplay experience. Curiously enough, to tackle the problem of changing illuminance, e.g. during the door opening, the lightmaps are layered with supplementary ones that are used to approximate the result.



Figure 2.19: Low-poly art style of Lara Croft GO is lightweight for mobile devices. Source: *Lara Croft GO*, 2015

■ GO series

The GO series of video games by Square Enix, namely Hitman GO, Lara Croft Go, and Deus Ex GO was initially developed for mobile devices in mind, therefore the graphics of those titles is simple by nature. Each title has its distinct style, with Hitman having the look imitating board games with plastic figures, Lara Croft having low-polygon look with flat surface colors and Deus Ex being somewhere in the middle of the road, emphasizing more on effects such as reflections, bloom, and particles. Arguably most intriguing is the graphics style of Lara Croft title that reminds of *The Witness* game, with its design boiled down to the essence. Lara Croft GO has its way in balancing rough and soft edges or geometry details in contrast to flat surfaces, making the complete title very lightweight both in style and performance [Mon].

■ Life Is Strange

Life is Strange tackled the non-realistic design with one idea in mind: To make the game look like a concept artwork [Cap16]. Object shapes are modeled realistically, the lighting is plain, but what makes the game stand out is the texture design. Each texture was manually painted, effectively rendering the unique style.



Figure 2.20: Life Is Strange’s art design tries to capture the essence of concept artwork, where each texture is hand-drawn (observe the jeans in detail). Source: *Life Is Strange*, 2015

■ The Walking Dead



Figure 2.21: The Walking Dead video game imitates the comics original by relying mostly on texture stylization. Source: *The Walking Dead*, 2012

The Walking Dead game series seems like another case of cel-shading, similar to Borderlands, as the static images of both games seem very close in terms of stylization. Oddly enough, The Walking Dead shares more similarities with Life is Strange graphics. The Walking Dead is based on a

comic book series, hence the stylization. Omitting later installments of the series, the original design was mostly built on plain textures drawn in a form of their comic roots. It is nevertheless a curious case of playing on the players' anticipation since at the end of the day the game's art seems very familiar, yet off from what those players are accustomed to.

2.3.2 Games imitating established art style

Ōkami



Figure 2.22: Ōkami draws visual inspiration in Japanese ink wash painting, though the result is "crowded" in comparison with original art style works. Source: *Ōkami*, 2006

Ōkami goes the distance to exhibit oriental ink wash painting known as *sumi-e* as an interactive visual style. The very idea is bold, as *sumi-e* art actively follows strictly minimal representation: Objects represented in *sumi-e* art are merely shortcuts capturing the essence of the real world objects with as little ink as possible [Mam12]. For this reason any attempts to bring the style to animated work have been very sparse and often distant from the original paintings. Ōkami delivers on this premise using the well-known cel-shading technique to create packed virtual world. The result is clearly reminiscent of the original *sumi-e* style, though lack of depth in technical section to appropriate the style drags the result to an artistically more general one.



Figure 2.23: Old ink wash painting. This traditional art style served as a model for both *Ōkami* and *Assassin's Creed Chronicles: China* video games. The final products differ immensely. Source: *Katsushika Hokusai, c. 1848*

■ Assassin's Creed Chronicles

Assassin's Creed Chronicles series adapts various designs through its installments. Despite being hard to classify as adapting one general art style, China installment, in particular, draws strong inspiration from sumi-e style and oriental painting across the board. It makes up for a good comparison with *Ōkami* as both games follow the same visual idea. Unlike *Ōkami* though, *Assassin's Creed* uses bits and pieces of the style to visualize only certain parts of the environment and props; foliage and textile are usually fabricated in this manner, creating partly transparent objects as spilled ink spots. The game doesn't force the matter, producing a visual style that does not make compromises for getting "as close" to some visual style.

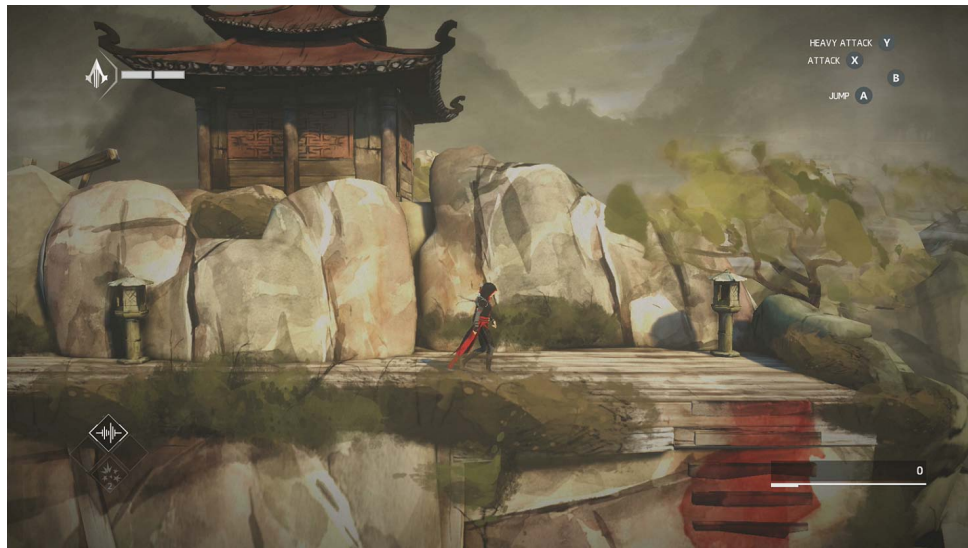


Figure 2.24: Similarly to *Ōkami*, *Assassin's Creed Chronicles: China* built its visual to resemble oriental ink wash paintings, though just enough to create own identity. Source: *Assassin's Creed Chronicles: China*, 2015



Figure 2.25: The visual style imitation of *Dreams* is not unified, however the art styles are often traditional. Source: *Dreams*, 2019

■ Dreams

Much like the previous series, the *Dreams* game implements techniques as a rendition of traditional visual styles, though without particular denomination, mostly reminding of aquarelle paintings. The environment and objects within are heterogeneous in terms of graphical presentations; the game characters often adapting image of plastic figures or stuffed dolls.

Valkyria Chronicles



Figure 2.26: Valkyria Chronicles series aims to combine watercolor paintings with pencil drawings. Source: *Valkyria Chronicles Remastered*, 2016

Valkyria Chronicles is a series that found its image in several art techniques; notably watercolor painting, which marks indistinct outlines and color bleeding without affecting the clarity of the design. Additionally, pencil drawing technique and likeness is applied. This is most prominent in backgrounds or shadows, that are crosshatched, smoothly blending with the watercolor basis.

Ni No Kuni

Ni No Kuni series imitates Japanese anime art style [Ric18], rendering its position favorable, for this particular style of animation builds upon simple imagery. The game itself uses cel-shading style and, visually speaking, achieves its goal. Still images from the game are unrecognizable from anime stills. This combination speaks in the game odds since cel-shading style is nowadays often used in anime cinematography. The style usage in the game is not punctual however, and the environment and animation do not carry the anime illusion to the motion.



Figure 2.27: Ni No Kuni series imitates Japanese anime, which is convenient for its art simplicity. The game makes due with simpler cel-shading method. Source: *Ni no Kuni II: Revenant Kingdom*, 2018

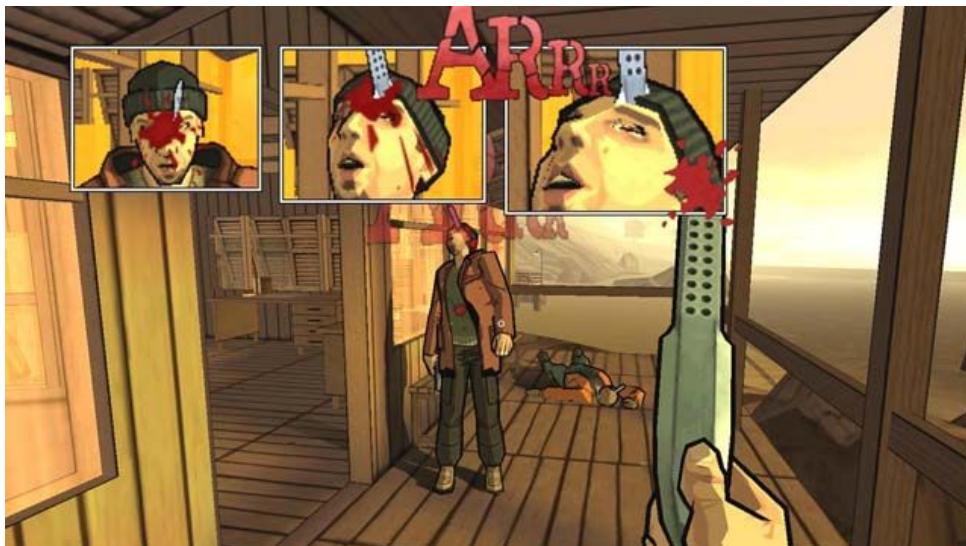


Figure 2.28: XIII was released in 2003 and was part of a first wave of cel-shaded video games. The style was the main reason for its prominence. Source: *XIII*, 2003

■ XIII

XIII finds itself in a similar position as Ni No Kuni, where the source material for the game comes from a comic book series. Thus the game imitates the art style of comic books, where the simplistic nature of cel-shaded graphics works well, applied to realistic models with rough shading. The game even

features classical comic book onomatopoeic effects, that visually spell the sound effects occurring in place.

■ 11-11: Memories Retold

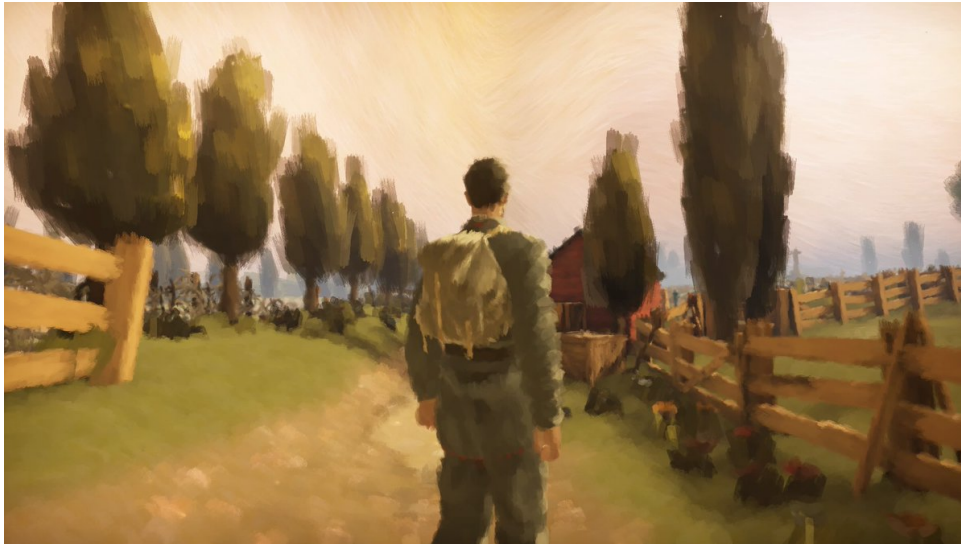


Figure 2.29: Technically, 11-11: Memories Retold is possibly the most successful representative of watercolor style-imitating video game. Source: *11-11: Memories Retold*, 2018

11-11: Memories Retold is an ambitious project co-developed with Aardman Studios, best known for stop motion clay animated movies Wallace and Gromit or Shaun the Sheep. The game pursuits impressionistic visual style [Ami18]. The design is not complicated and the images are homogenous: Without too much detail, the image is rendered with even brush strokes, controlled primarily by the stroke width and direction alongside possible adjustments to the coherence and life-cycle of the strokes. The result is a modernistic looking game imitating works of artists such as Turner and Monet.

■ 2.3.3 Games imitating realistic craft

Listing games imitating art style, games presented above endeavor imitating either drawn or painted style, while other games were omitted. This comes from the nature of rendering mechanics, since games imitating, say sculpting techniques imitate plain real-life objects. To put it another way, those games are expected to provide a realistic rendering of those objects, thus marking them as using non-realistic graphics techniques is technically incorrect. Of

course, they still apply for the list of games imitating art styles, so to complete the list in a meaningful sense, few examples of games done in particular crafting manner are presented. All of the games listed below come from Nintendo company, thanks to its devotion to artistically experiment with virtually any of its titles.

■ Kirby and the Rainbow Curse



Figure 2.30: Kirby and the Rainbow Curse produced a full-fledged clay environment. Source: *Kirby and the Rainbow Curse*, 2015

Kirby and the Rainbow Curse was done in claymation art style [Vog15], imitating the look of polymer clay. Using such style is not particularly unique; what makes it original, however, is the fact the style is purely imitated, whereas claymation has its place in games usually backed with real life stop motion clay animation, for instance in titles Armikrog or The Neverhood. To come as close to the original art style as possible, Kirby game uses methods such as lowering the framerate to leave the game with signature stop motion feel.

■ Yoshi's Woolly World

Yoshi's Woolly World features characters taking shapes of yarn knitted dolls. Environment and game objects are also either woolen knits or props made of fabric, leather, plastic or various other materials usually used in home do-it-yourself craftship.



Figure 2.31: Yoshi's Woolly World crafts most of its world from yarn. This design is almost unique. Source: *Yoshi's Woolly World*, 2015

■ Paper Mario: Color Splash

Paper Mario: Color Splash is one of the entries in the long-running Paper Mario series. The series presents characters as paper cutouts, where the Color Splash title came technically furthest, which allowed for almost every environment and game object to be virtually crafted out of paper with high fidelity [Far16]. This attitude is similar to Yoshi's Woolly World, whereas the material is different obviously. The game imitates various types of paper, alters its weight and wear, and builds the scene realistically, which results in details like cut out clouds hanging from threads or background folds and seams being visible. This game, in particular, is a great argument in a discussion about categorizing games imitating realistic art crafts, as considered above. Color Splash involves random real-life objects such as lemons or fire extinguishers, or even complete common life environments e.g. a kitchen, rendered without any stylization, photorealistically. The paper world doesn't shatter the overall image, hence the technicality of stylization comes from the craft chosen, not the non-realistic rendering.



Figure 2.32: Paper Mario: Color Splash crafts every object from virtual paper, including sheets of paper as backgrounds. Source: *Paper Mario: Color Splash*, 2016

2.3.4 Goals

The vision for non-realistic graphics in games is everpresent, regardless of the quality of this direction in each respective game. Taking an example of *League of Legends*, the game is already stylized in toon graphics, while the design of supplementary graphical media is even more innovative. It benefits from suppressed interactivity of such media, for instance, the login screens are paintings brought to motion by layering parts of illustrations [Bro], which in turn allows for the image to gain a slight possibility of movement in certain points, affecting attached areas. To name another addition to the games media, a short movie called *ANNIE: Origins* was introduced to further develop the lore of the game [RG18]. The movie was put together from an initial state of 3D scenes stylized in painterly design. Afterward, each frame gained further work and details manually, getting closer to the idea of the movie being hand painted completely. The result may mark a milestone for certain games, where those games virtually deny the 3D object nature, leaving only the 2D impression, without the need of manual revision.

Ideally, games like *The Banner Saga*, which is played in isometric view with characters hand drawn, should be indistinguishable in terms of art and animation from games rendered using non-realistic graphics techniques. At the same time, it is necessary to avoid a “rotoscoping syndrome”, meaning the animation carries over the structural quality of three dimensions, which is usually undesirable, as it gives away the frames are merely redrawn from the 3D foundations, as observable in films made with the rotoscoping technique.



(a) : Login screen animated by layering multiple paintings.



(b) : Short film Annie, prepared in 3D and finalized manually.

Figure 2.33: Supplementary visuals for League of Legends video game.

It is worth noting that software solutions for artist is, as per usual, ahead of consumer products. Naming one, the *Rebelle* software by *Escape Motions* is a showcase of drawing and painting capabilities, should one resort to the computer as a drawing and painting tool [Bla16]. *Rebelle* is very much physical simulation of artistic tools and paints reacting with paper foundations, imitating techniques as paint splatters, blending or drying with textures, to list just a few. It is safe to proclaim works made with this software are in instances truly indistinguishable to artworks made with real-life tools, thus crossing the threshold the video game industry is still waiting for.



Figure 2.34: A painting produced with Rebelle software. Source: *Philipp Neundorf, 2018*

Chapter 3

Related works

It is apparent from the presented overview of video games there are many various approaches toward the non-photorealistic rendering. The primal difference of those comes from the art style demand - how versatile the method could be: It is arguably less complicated to imitate some particular art style, rather than to create a unified solution for arbitrary stylization, due to the possibility to concentrate effort on singular issues of the style.

3.1 Specific art style imitation

Montesdeoca et al. presented multiple works [MSR16, MSRB17, MSB⁺17] on imitating the watercolor painting style. They propose a system [MSR16] which divides the inclusive method into a routine of simulating singular effects occurring during the watercolor painting, using a number of shaders in its rendering pipeline to apply individual effects.

Those effects are divided into categories with regard to the input data: Image-space simulations and Object-space simulations. Object-space simulations process the scene on a scale of objects, each object being susceptible to some local stylization, whereas Image-space simulations shape the stylization of an image as a whole, which is necessary for simulating effects like bleeding, that occur in conjunction of multiple objects. Both categories of simulations complement each other and are not to be separated.



Figure 3.1: A particular art style imitation: Watercolor paintings. An output of Montesdeoca et al. methods. Source [MSB⁺17]

Starting with Object-space simulations on a vertex shader, the transformation of vertices from object space to the projection space is deformed by “Wet-in-Wet” and “Hand Tremor” deformations: The former allows simulating an effect of bleeding, thus spreading the pigment put on already wet areas outside of the placement areas, by translating vertices outside the original geometry; the latter allows for creating jitter edges caused by involuntary tremor from the nervous system, which is achieved by slightly offsetting the vertices of objects, once they are transformed to the projection space. Other methods deal with a dilution of the pigment, that makes up for the characteristic translucency found in the artwork; cangiante, which describes a change in color hue on areas of shifting brightness; or pigment turbulence, an uneven distribution of the pigment on given areas accommodated by a noise of low frequency.

Moving to the Image-space simulations, the system uses additional textures, the paper textures, and its normal map, and a low-pass filtered color image, altogether paving the way for additional effects: The color bleeding effect is applied to ensure the pigment diffuses on a paper foundation. This step is followed by edge darkening, which simulates pigment accumulation on the edges of colored areas caused by surface tension, implemented with *Difference of Gaussians* algorithm for feature enhancements. The final step considers the physical structural quality of paper foundations, causing effects of distortion and granulation that further affect the scatter of pigment.

Without additional setting, the result of these steps comes very artificially, without human imperfections and entropy bound to the art. Thus, an additional layer - a control image is introduced, further affecting local alterations of the outcome. The control concept is handled by simple imagery, with each

one of RGBA channels disjointed into affecting different simulation effects: Red channel controls paper distortion, green controls paper granulation, blue controls edge darkening and color bleeding, and alpha controls pigment turbulence and pigment fading. This concept finally ensures the imperfections that make the result feel human-conceived. Of course, the downside is this is partially true, as the control image conception is supposedly exercised by the artist using the system.

In a similar manner, the subject is further developed adding more singular simulations, e.g. [MSB⁺17] elaborates on perfecting the artwork edges and advanced occurrences in pigment reactions.

The issue with this manner of stylization, where singular simulations are applied, is the bond to each particular simulation the result is dependent upon, whilst it is dubious to state the steps needed to enhance the result. A method that addresses the granulation of simulations, may be based upon using predefined patterns or generalized “brush” strokes to synthesize the result.

3.2 Stylization based on contained patterns

Following in the vein for listed video game art styles, there are many algorithms that focus on imitating oriental paintings and art style, using a predefined set of patterns and brush strokes, couple of those is listed below to provide an insight into this type of imitation methods.

[MTD04] proposed a system, which synthesizes the brush strokes used in Chinese calligraphy and painting. This work may be perceived as an evolutionary step that led to the development of techniques like [WLS02].

The trees synthesized with the stated algorithm are processed on a texture level, where every generated tree silhouette is subjected to determining various aspects: Orientation of the texture, its distribution, dependency on view, and ways of preserving the painting-view. The algorithm generates various reference maps, analyzing the geometry and shading, consequently generating the tree textures and mapping them within the imagery. Despite being able to adjust the technique for different types of tree, varying both in shape and



Figure 3.2: A horse painting created with simulated calligraphy brushes. Source [MTD04]



Figure 3.3: A result of 3D trees processed with method for imitating Chinese paintings. Source [WLS02]

the brush strokes nature, the system is so specific it epitomizes the general issue with systems build upon synthesizing contained art techniques: The

output is limited by the spectrum of defined patterns, or brush strokes in this case, leading to very low versatility of the system.

That shifts the topic that leads to imitation type used in this work.

■ 3.3 Example-based stylization

Taking prepared artworks, it is possible to create a system, which is not dependent on any particular art technique, and at the same time being able to imitate idiosyncrasies of the given style. Such techniques establish the stylization on a provided example artwork.

■ 3.3.1 StyLit

StyLit algorithm introduced in [FJL⁺16] yields great results in synthesizing imagery based on provided examples. The algorithm itself is based on illumination guidance and the input is a triplet - user provides a model artwork, which serves as an imitation basis; the same model is rendered as Light Path Expression (LPE), a technique used to separate different illumination effects into isolated components, e.g. direct diffuse render, direct specular render, etc; the final input comes from a target object render, rendered as the very same LPE. This means both source input images are aligned, featuring the same scene in a sense of geometry and placement.

The algorithm runs several iterations, each of those carries over patches of the source style artwork and applies them into the target to be synthesized. The process is guided by the aforementioned images of LPE, where the best-matching patch for each respective region is chosen accordingly to the local similarities of the source model and target images.

It is intended for the patch usage to be uniform, where an often occurring issue emerges: Distribution and size of regions of similar illumination on both the source model and the target are likely different. Thus, to meet the uniformity constraint, the algorithm would enforce patches that are inappropriate for some areas of the target. This is resolved by calculating the error “budget”, that results in constraining the feasible solution. Running



Figure 3.4: An arbitrary art style imitation achieved by the StyLit algorithm. Left: Source style and model. Right: Result image. Source [FJL⁺16]

the algorithm in several instances, as mentioned, initially creates a coarse result, which is further fine-tuned to the optimal result.

The method provides great outputs for arbitrary stylization, imitating every part of the source style imagery, objects, shadows, and backgrounds. While unsuitable for real-time application, the system is a starting point for many other following algorithms.

■ 3.3.2 Expressive Animation of 2D Rigid Bodies

[DBB⁺17] introduced a system using example-based stylization for animating 2D rigid bodies. The system is capable of replicating motion style, transferring it to given new animation defined by a series of “rigid” frames. The stylization impacts the visual style the same as the characteristics of motion, protruding in deformities of the object.



Figure 3.5: An imitation of a visual and motion style in an animated sequence. Left: Target sequence. Right: Resulting stylization. Source [DBB⁺17]

The algorithm draws inspiration from classical 2D animation, which is usually crafted using keyframing: A set of frames defining the points of parametric change in a film, e.g. timing, where the frames in-between keyframes represent a routine change for a smooth transition. The input of the algorithm consists of a set of animation frames as a reference of the source rigid animation, a set of corresponding stylized animation frames, and a set of frames as a target for stylized animation. The algorithm trickles-down the hierarchy of frames scaled for each level: key events level, pose to pose level, and frame to frame level as a final animation level. Through decomposing the source samples into geometric deformations, the system synthesizes parametric deformations, enforcing them to the target, while the visual appearance imitation is handled by StyLit algorithm from [FJL⁺16].

3.3.3 ToonSynth

Similar subject to the previous work may be found in [DLGKS18]. The task formulated there is to transfer 2D animation, both its visual and motion style, given existing stylized sequence and respective skeletal animation, to a new animation sequence defined by a new skeletal animation. Again, the algorithm combines StyLit [FJL⁺16] algorithm for visual style transfer, with a method to capture and replicate the motion characteristics.

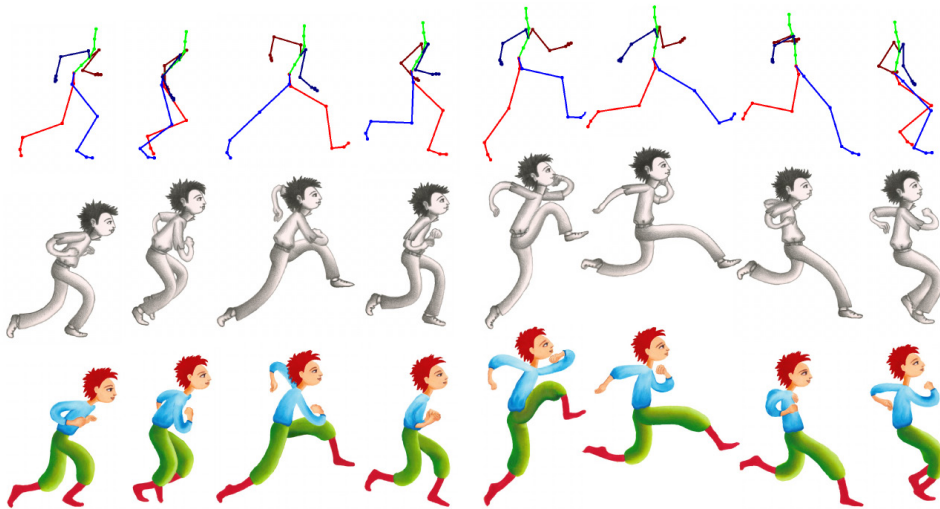


Figure 3.6: Two animation sequences produced by the ToonSynth algorithm, using the skeletal target sequence from the top row. Source [DLGKS18]

Animation subject provided as an input is required to offer its decomposition into layers, literal dismemberment of self. The method creates a style-aware puppet, consisting of the stylized layers - the body parts, connected at junction points. This allows for carrying out a procedure to register the deformation

of puppet parts. Obtaining a target frame puppet, the method searches for the closest-similarity sub-sequence of the source skeletal animation, where intersecting result sub-sequences are blended together for synthesising the output animation.

3.3.4 Facial animation

Another use of the technique StyLit [FJL⁺16] comes from a method for an example-based stylization of facial animations proposed in [FJS⁺17]. The method is able to synthesize stylized facial animation, provided the real-life footage of such, as well as style artwork image as an input. The complexity of such stylization is substantial since it requires translation of each source facial feature onto the corresponding target region, therefore the method analyzes and creates the facial structure guidance.



Figure 3.7: Art styles (top row) transfer to the target image of a face (bottom left). Source [FJS⁺17]

The guidance uses several channels to transfer the source patches (substituting the original StyLit guidance): Segmentation guide is generated as an image representing the subdivision of facial regions, e.g. eyes, mouth, hair, nose. Connecting to the Positional guide, this channel is a deformation field, propagating the source and target facial structure shift, effectively resulting in the source patches being transferred to similar relative positions in the target image. Appearance guide is a grayscale channel, that is used to maintain the facial identity, capturing the shading gradients of the target. As human perception is sensitive to the subtlest local differences in the course of facial recognition, the subjects identity perception may be easily severed. Thus, this guidance channel analyses the global intensity levels and local contrast values in the source image to recreate the same ambiance for the target. Finally, the Temporal guide deals with the balance of temporal coherence coming along

with the animation: While perfecting the coherence would seem wrong, as the hand-drawn animation lacks high-frequency coherence, to preserve the dynamics the coherence needs to be preserved at low frequency. The guidance channel uses blurred images of source style and previously rendered stylized frame, resulting in temporal flickering, which may be further controlled by the blurring kernel.

The system even takes into account issues connected to the facial animation that is opening and closing mouth and eyes, which need to be acknowledged and dealt with to avoid disturbing effects. Provided the guides, the system is able to recreate stylized facial animation generally indifferent to the art style provided.

3.3.5 Stylizing video

Similarly to the previous work, [JvST⁺19] introduced a stylization method for a video sequence, albeit unlike the previous one, it is possible to stylize arbitrary video sequence, should user provide stylized keyframes. In this case, the keyframe is a stylized video frame, prepared by an artist, which serves as a foundation for the stylization of additional frames. Input keyframes are not limited by quantity, on the contrary, the video sequence structure is not constrained and scene elements involved are variable, hence it may prove necessary to provide further stylized keyframes.

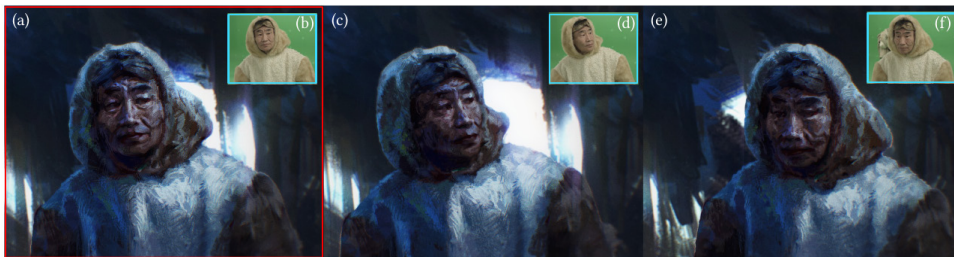


Figure 3.8: An original video sequence containing frames (b), (d) and (f) stylized using keyframe (a). Frames (c) and (e) are the products of the stylization. Source [JvST⁺19]

As another method using the technique of StyLit [FJL⁺16], this algorithm requires generating a set of guides to transfer the style patches. Those are semantically similar to the ones followed in [FJS⁺17]. Besides the input frames, they involve Mask guide, Positional guide, Edge guide, and Temporal guide. The Mask guide is not obligatory, as it encourages the stylization to respect the object boundaries and occlusions, although if the accuracy is not tight for the purpose of stylization, it may be omitted. The channel represents the

boundaries of the objects. Positional guide is used to maintain the structural quality of the scene, controlling the style transfer to appropriate regions. This channel uses coordinate mapping, where corresponding coordinates of keyframe propagate to their respective positions in the sequence frames, utilizing the motion field. Edge guide is generated using edge detection and is applied for the style to avoid losing sharpness around the edges. Temporal guide uses previous synthesized frames to maintain the coherence, where the currently synthesized frame is enforced to follow similar transfer procedure path as the previous ones, keeping the animation fluent.

As discussed, the method may be provided with a number of keyframes, thus creating an issue of potentially corrupting stylization cohesiveness. The method proposes a nontrivial solution of blending multiple synthesized frames of the same, each stylized accordingly to different keyframes.

This brings the focus to StyleBlit algorithm, an example-based technique, sharing similarities with previous works; the rest of this paper is devoted to this algorithm.



Chapter 4

StyleBlit



4.1 Overview

To translate any arbitrary art technique to video game environment is beyond feasibility potential of methods presented up to this point. Following chapter presents a method that is capable of providing stylized imagery required for video game environment at framerate required by the medium, and detailing the implementation of this method in Unity game engine.

StyleBlit, as proposed in [SJT⁺19], shares similarities with StyLit algorithm from [FJL⁺16], as described earlier, using guidance channels to appropriately transfer patches of original textures. As it turns out, using fine guidance and relying on the source style texture to be stochastic - absent of regularities, which usually comes along with hand-drawn samples, the textural coherence becomes less of a problem, since local guides implicitly encourage coherent placement, while stochastic nature of texture helps to mask the seams between patches.

This finding initiated the development of a solution, which relieves StyLit's endeavor for textural coherence, and, consequently, is able to provide real-time results.

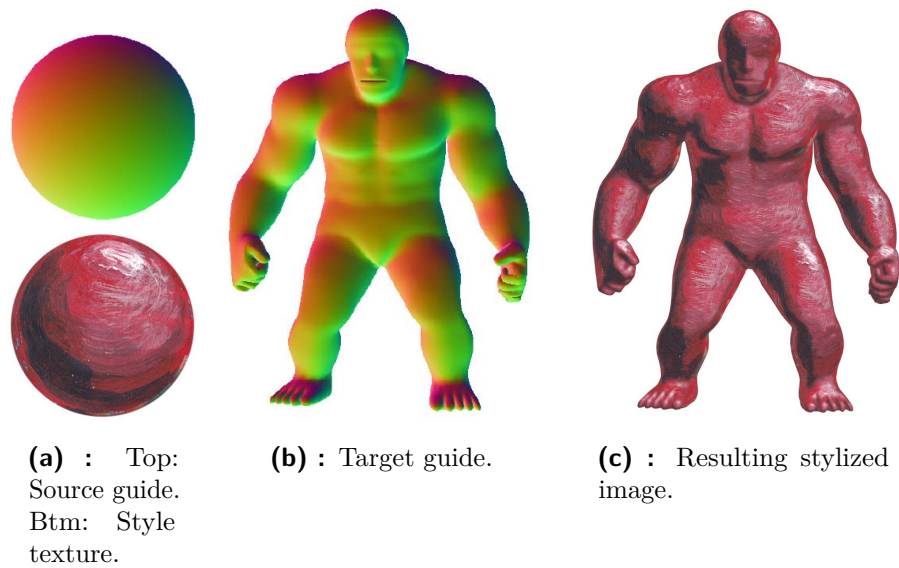


Figure 4.1: An overview of inputs required for stylization and a resulting output.

4.2 Algorithm

The method is designed to transfer chunks of the source style texture to the corresponding region on a target image. For better understanding of the algorithm, it is important to realize the pixels comprising each chunk is a 1:1 region of the source texture. That means the generic version of the algorithm processes an “anchor” point, with surrounding region pixels being relative to this point. Coordinating pixel placement is achieved by using local guidance system, i.e. normal maps, texture coordinates, etc. Whatever guide is used, the algorithm uses values of both source and target guides to spatially align the chunks:

The placement location is determined by either a processed pixel obtained in scan-line order or using a random seed in the target image. By calculating distance values between the source and target guides in local spatially-aligned regions around the processed point, it is determined whether the target region pixels belong in the same chunk by comparing said distance values with a given threshold distance value. If so, each corresponding source pixel is copied to the target image, eventually covering it whole after repeating the search and copying step.

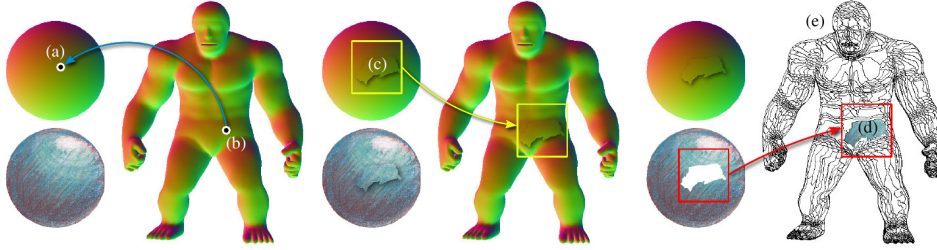


Figure 4.2: Visualization of the method: For each random seed (b) a corresponding position in a source exemplar is located according to the guidance (normal) value (a). For each corresponding target and source pixel pair in a spatially aligned region around the seed, their guidance values are compared. If the guidance value difference is below the threshold, the pixel is set in the same chunk (c). The chunk is transferred to the target (d), where repeating the process fills the whole target mosaic (e). Source: [SJT⁺19]

Algorithm 1: Brute force StyleBlit

Inputs: source style exemplar C_S , source guides G_S ,
target guides G_T , threshold t .

Output: target stylized image C_T .

Function styleblit(C_S, G_S, G_T, t):

```

foreach each pixel  $p \in C_T$  do
  if  $C_T[p]$  is empty then
     $u^* = \operatorname{argmin}_u \|G_T[p] - G_S[u]\|$ 
    foreach each pixel  $q \in C_S$  do
      if  $C_T[p + (q - u^*)]$  is empty then
         $e = \|G_T[p + (q - u^*)] - G_S[q]\|$ 
        if  $e < t$  then
           $C_T[p + (q - u^*)] = C_S[q]$ 
  return  $C_T$ 

```

The brute force version of the algorithm from Algorithm 1 demonstrates the straightforward and expressive approach, though it is understandably inadequate in terms of performance, making up for an easy interpretation. The parallel version (using seed grids) is elaborated on in details in the following section concerning implementation.

To accelerate the step of retrieving the source texture pixel corresponding to the guidance value, it is vital to use either some look-up table, suitable should the guidance be simple, or search trees, should the guidance be advanced.

An additional (voting) step is applicable for smoothing out seams or suppressing scant noise by using linear blending.

4.3 Implementation

The implementation utilizes Parallel StyleBlit algorithm version presented in Algorithm 2. This version of the algorithm realizes suggested seed hierarchy: A grid of implicit seeds is defined - their distance decreased by a factor of 2 on each level, while their position is randomly displaced. Stretching from the algorithm, this effectively means each chunk is subjected to appropriate seed; for any processed pixel, the seed search is performed, which either satisfies the distance constraint, in which case the source pixel is transferred, or the search resumes on a lower level, making for maximizing the chunk sizes.

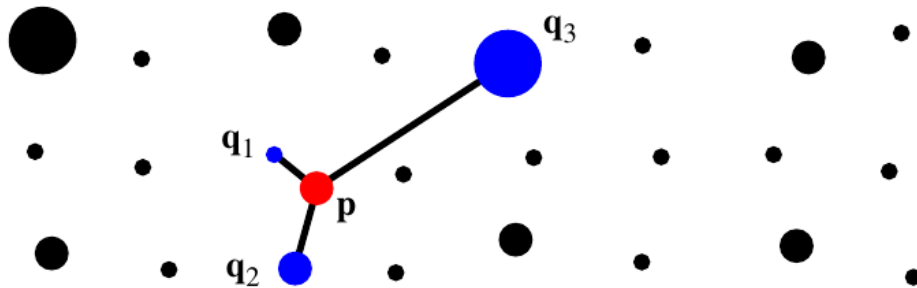


Figure 4.3: The seed hierarchy represented by blue and black points. For a target pixel p (in red), the algorithm descends the seed grid levels (visualized by dot size), where the space between seeds decreases with each lower level. In this case the pixel p first finds the closest top level seed q_3 . If the guidance value between the two is higher than the threshold, then the search continues on q_2 and possibly q_1 , until an appropriate fit is found. Source: [SJT⁺19]

The Unity implementation consists of a main shader of three passes written in HLSL and a supplementary C# script.

The implementation uses normal maps as guides for style transfer, hence it is required to obtain a normal map for each object. This is an objective of the first pass of the shader. Normal values are retained invariant to camera

position, thus their spectrum remains the same, aligned to view space. The normal map is rendered for another pass in a form of texture.

The second pass is an essential implementation of the Algorithm 2. To picture the process, the seed hierarchy in relation to the rendered normal map is virtually a sampler of map's normal values comprised of different region sizes and shapes, funding the chunks the algorithm resolves to. Each target pixel undergoes the search for nearest seed, where the guidance error is calculated to determine whether the target pixel is fit to belong to the seed-defined region, hence whether the corresponding source pixel is fit to be copied into target place. The implementation utilizes suggested look-up table for translating normal values to corresponding coordinates of the source style image, thus the final step might very well involve simple translation of said coordinate between the source style texture, albeit to smoothen the result, suppress the seams and stranded noise pixels, an another pass is applied to realize the aforementioned voting step:

Coordinate values are not straight up translated; they are carried over to the final pass as a texture. This pass is fashioned to average the neighboring pixel values before the final translation and copying of the corresponding pixel.

The supplementary script has a couple of functions: Firstly, it implements a method, which renders the look-up table, mapping coordinates of style texture pixels into red and green color channels of the texture. Those coordinates are positioned accordingly to red and green channel values of the source guidance texture. This process allows simple retrieval of source texture pixel for an arbitrary normal value that is encountered, simply by using the normal values as texture coordinates, where returned values are used as UV coordinates to retrieve the source texture pixel.

Additionally, the script renders random noise texture, which is utilized in the seed search. The use of jitter values affects the “shape” of the seed region, meaning the perpetual jitter changes the spatial characteristics of the chunks comprising the final image. This results in adjustable loss of temporal coherence, important for the imagery to look like hand-drawn.

The script further augments the temporal coherence, limiting the framerate and noise texture rendering to approach the ideal of drawn animation.

Algorithm 2: Parallel StyleBlit

Inputs: target pixel p , source style exemplar C_S , source guides G_S , target guides G_T , threshold t , number of levels L .

Output: stylized target pixel color $C_T[p]$.

Function SeedPoint (*pixel* p , *seed spacing* h):

```

|  $b = \lfloor \frac{p}{h} \rfloor$ 
|  $j = \text{RandomJitterTable}[b]$ 
| return  $\lfloor h(b + j) \rfloor$ 

```

Function NearestSeed (*pixel* p , *seed spacing* h):

```

|  $d = \text{inf}$ 
| for  $x \in \{-1, 0, +1\}$  do
|   | for  $y \in \{-1, 0, +1\}$  do
|     |  $s = \text{SeedPoint}(p + h(x, y), h)$ 
|     |  $d = \|s - p\|$ 
|     | if  $d < d^*$  then
|       |  $s^* = s$ 
|       |  $d^* = d$ 
| return  $s^*$ 

```

Function ParallelStyleBlit (*pixel* p):

```

| foreach level  $l \in L\{n, \dots, 1\}$  do
|   |  $q_l = \text{NearestSeed}(p, 2^l)$ 
|   |  $u^* = \text{argmin}_u \|G_T[q_l] - G_S[u]\|$ 
|   |  $e = \|G_T[p] - G_S[u^* + (p - q_l)]\|$ 
|   | if  $e < t$  then
|     |  $C_T[p] = C_S[u^* + (p - q_l)]$ 
|     | break
| return

```

4.4 Unity

With an additional layer of a software environment in the form of a game engine, a middleware, in reality, the development is limited to the extent of the graphical possibilities of the engine. Therefore, as may be expected, the implementation utilizes some specific solutions for the game engine environment of Unity.



Figure 4.4: The first pass output: A Unity-generated normal target guide.

First and foremost, the issue which projected into many parts of the implementation process was the very effectivity and potential of a Unity multi-pass shader, which is crucial for the implementation. In an initial version of the plugin, much effort was put into creating a workaround for the multipass implementation. The reason is twofold: The multi-pass, as executed by Unity, raises quality problems with very limited control over its state. The pass output texture rendered for the following pass suffers a quality loss that is almost unnoticeable, however, provided with coordinate values bound into the color channels of the texture, the coordinate values must preserve its quality. If this texture's pixels are translated into the source style imagery, the image exhibits strong quality loss, rendering the result worthless. The second reason comes from the technical ineffectivity of the multipass execution, where the pass texture captures much of the screen view, merely complicating any process localizing pixel neighborhood, such as the second pass of the implementation. Eventually, the performance of the multi-pass might become a concern.



Figure 4.5: An early iteration of a sample chunk distribution.

Unity offers a method to render objects into texture, which enables one to obtain textures usable for their respective shaders by adjusting a camera and a special texture. This process is suggestable as a substitute for some multi-pass shaders since it offers much higher control of the process and the quality. After running many tests with this technique, adjusting the pipeline accordingly, it turned out the adjustments and settings of the scene tools and cameras are overly complicated, amplified by the fact that the whole *StyleBlit* plugin for Unity is required to be as compact as possible, allowing for easy installation and usage. This effort was simply unfeasible and it panned out any adjustments to solve the stated issues of the multi-pass solution are preferable over the rising issues of rendering to texture.

Another notion to bear in mind during the implementation was the method's indifference towards the stylized imagery scale, or, put from the opposite perspective, the source style texture patches maintaining their scale. For the Unity environment, this means calculating screen space positions and gathering additional texture parameters.

Furthermore, the implementation made ready for video games revealed several challenges in the gameplay rank. One of the typical problems in first-person-viewed games surges with camera angle being too wide. This causes stretching of the objects on screen edges, which, in combination with the stylization method, creates repetitive maps on the objects, as the stretching leads to the guidance channel to exhibit little to no shift, technically acting similar to a flat surface. This was addressed in the implementation by adjusting the camera angle accordingly, although should the gameplay require it, the problem may ask for specific treatment. Similar challenge attached to the gameplay is directed towards the framerate capping. It was already stated



(a) : A visual representation of the second pass output: A texture of coordinates for retrieving the source texture pixel.

(b) : An early coarse result of pixel retrieval from the texture coordinates, without the voting step.

Figure 4.6: A visualized result of the second pass, before and after the source pixel retrieval.

the implementation caps the framerate so that imagery resembles hand-drawn animation. Clearly, there is a discord between the framerate scored and required by actual gameplay, and the framerate common for hand-drawn animation; the former usually scoring at least 30 frames per second (fps), the latter at least 12 fps. As became evident from the tests ran through the implementation phase, the cap settings must reflect the level of control over the camera. A free camera-control game, such as a shooter, may drop the 30 fps requirement very tightly, whereas camera-static game might prove viable in as low as 12 fps. However, the implementation exploits the dichotomy of actual animation and the stylization process: Providing the animation frames at, say 24 fps (the implementation caps fps at that rate as well), it is sufficient to render the noise texture on every other frame, i.e. the stylization property changes at 12 fps. This process retains the dynamic clarity of a scene while imitating hand-drawn quality. Naturally, it is not necessary to render noise texture for each object individually, in fact, this may cause performance issues.

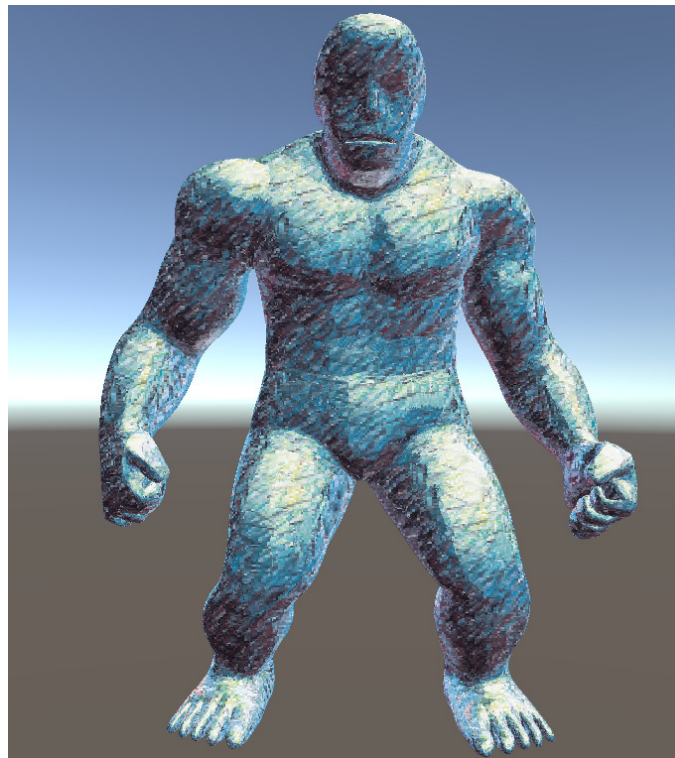


Figure 4.7: An early iteration of a resulting image from the third, voting pass. Notice the quality degradation caused by the multipass.

4.5 Results and evaluation

The final plugin was evaluated with respect to various aspects: Performance, practical usage, aesthetics, and compared to the vanilla version of StyleBlit, produced as a showcase for the original project.



Figure 4.8: First stylized animated exemplars produced by the plugin. Tiger model source: *Zealous Interactive*, iguana model source: *Junnichi Suko*

4.5.1 Performance

For the purpose of performance testing, the key was to create a benchmark project, which simulates a video game environment, and at the same time provides a technical challenge. Thus, the testing employed a Unity multi-material animated asset by *3DMAesen* and was conducted on three computing configurations. The scene consisted of approximately 12000 triangles. Fps capping was obviously disabled. The scene pictures a fighting knight, as seen of figure 4.9.



Figure 4.9: A performance test scene capture. Model source: *3DMAesen*

The tests were run with different quality levels set by Unity itself (version 2019.1.1). The major trait confirms it is possible to create a full game scene using the StyleBlit stylization, thus creating a full-fledged video game using this method. What may come as a surprise is the trend following the quality scaling: The framerate increase step does not evenly follow the quality settings level regression. Oddly enough, the framerate periodically decreases with a lower quality level. This phenomenon reflects the fact that this graphical stylization lacks advanced graphical effects, let alone lighting issue in general. The primal quality settings impact then resides within texture quality, filtering, and anti-aliasing method and quality.

In conclusion, the results data demonstrate it is possible to play a video game utilizing StyleBlit on a gaming computer with as high as 60 fps and more.

Res./Fps	Ultra	Very high	High	Medium	Low	Very low
1366 x 768	29	30	37	42	42	43
1920 x 1080	19	20	24	28	29	29

Table 4.1: Performance test on CPU Intel Core i5-6300HQ 2.30 GHz with GPU Intel HD Graphics 530.

Res./Fps	Ultra	Very high	High	Medium	Low	Very low
1366 x 768	146	125	148	147	138	147
1920 x 1080	96	93	132	133	136	136

Table 4.2: Performance test on CPU Intel Core i5-6300HQ 2.30 GHz with GPU 2GB GeForce GTX 950M.

Res./Fps	Ultra	Very high	High	Medium	Low	Very low
1920 x 1080	379	378	460	521	563	550
2550 x 1440	259	260	318	382	386	386

Table 4.3: Performance test on CPU Intel Core i5-7600K 3.80 GHz with GPU 6GB GeForce GTX 1060.

4.5.2 Comparison with vanilla version

The vanilla version of StyleBlit program visualizes a stylized image of golem figure, therefore the same model was used in Unity testing, along with the same selection of style textures. This makes a direct comparison available, see figure 4.10. As evident from the images, both versions are visually corresponding, the prime difference being the smooth edge-to-background transition. This shortcoming on the Unity plugin side appears in edge transition between non-stylized objects and environment, however overlapping stylized objects exhibit smooth transition.

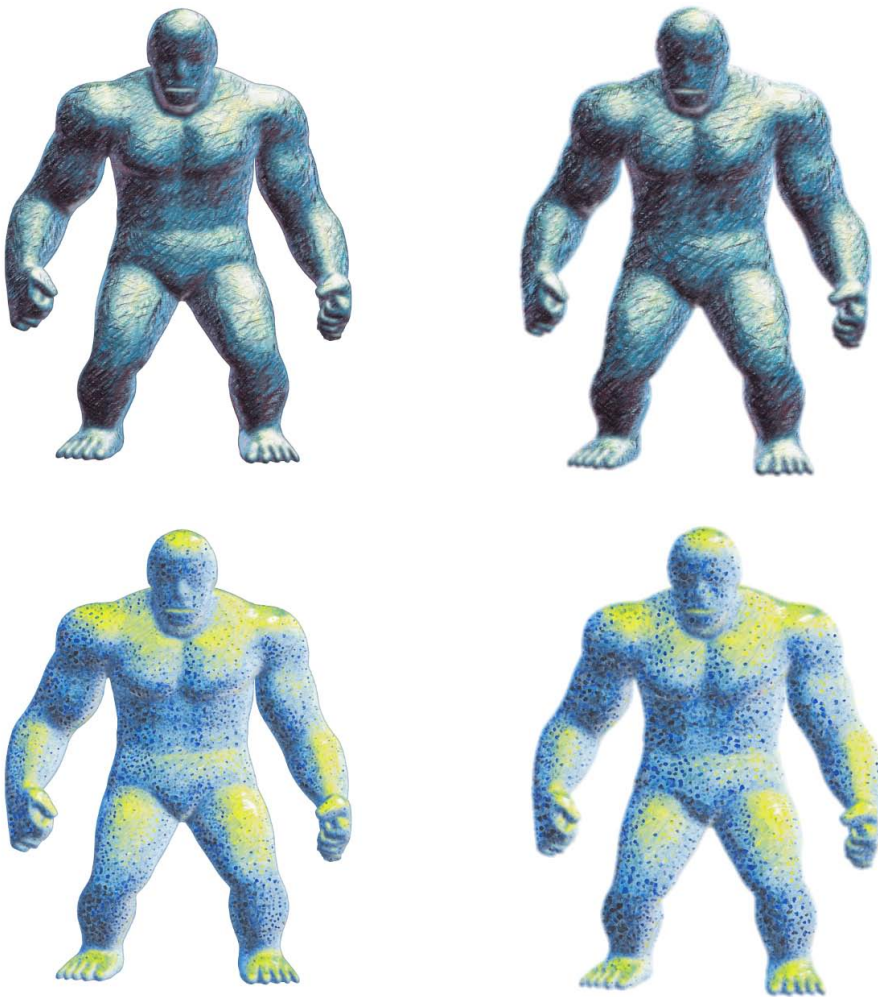


Figure 4.10a: StyleBlit plugin comparison with vanilla version. Left: Unity, Right: Vanilla



Figure 4.10b: StyleBlit plugin comparison with vanilla version. Left: Unity, Right: Vanilla

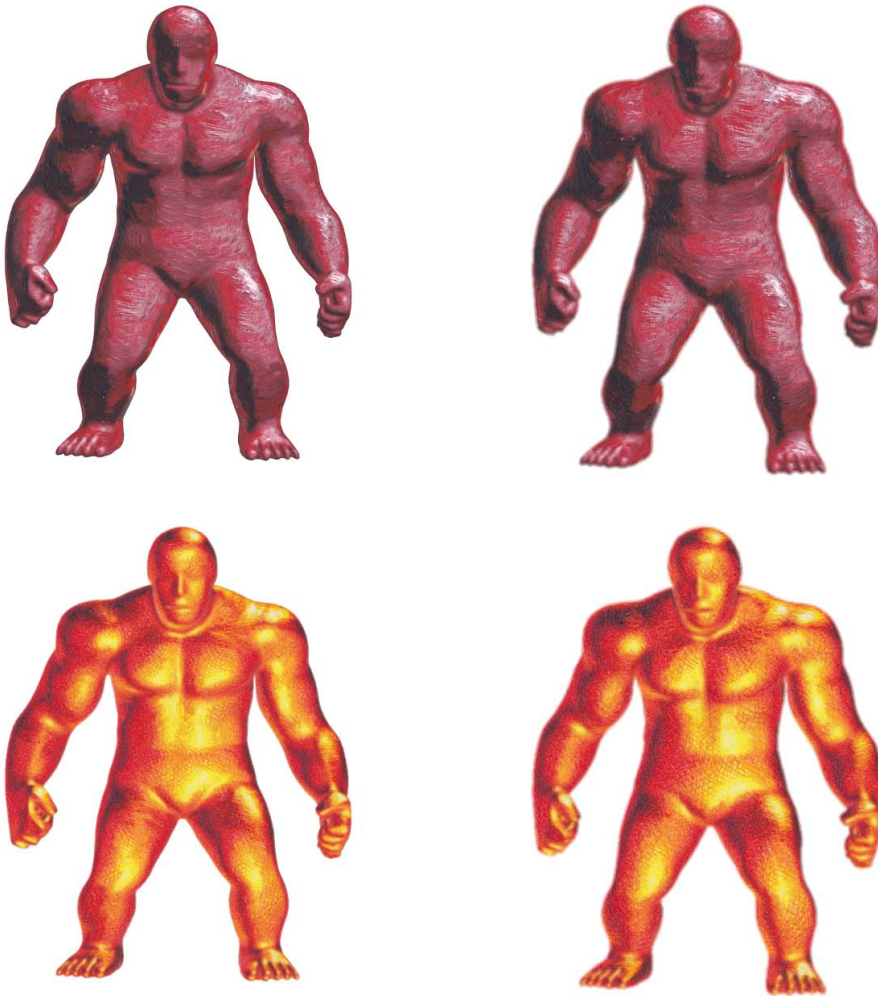


Figure 4.10c: StyleBlit plugin comparison with vanilla version. Left: Unity, Right: Vanilla

4.5.3 Practical evaluation



Figure 4.11: A watercolor stylization of a complex model with matching art environment. Model source: *Unity Technologies Japan*

The plugin was evaluated in game development: A point-and-click adventure game viewed from an isometric point of view. This is fitting settings for a stylization - the amount of scene objects is relatively low and the camera is mostly fixed, though this creates a stylization challenge for the environment. Rather than creating a specific solution, for instance, the one seen as on figure 4.11, where the environment is established by style-similar images, the development team decided to go with the fully material-stylized environment, utilizing StyleBlit on each environmental object. This might feel confusing or distracting during the user experience, for stylized objects flicker, creating an illusion of hand-drawn animation in process, hence a constant flickering of the whole scene might become tiresome for the player. Noted by the development team was a suggestion the StyleBlit may be very well used on objects with flat surfaces, should it be provided with appropriate style texture. This suggestion stems from the realization that even though StyleBlit's usage on monotonous surfaces is tricky, the stylization process ensures uniform coverage of the object with the style texture patterns. The small video game was successfully produced, figure 4.12 shows a sample scene from the game.



Figure 4.12: A screenshot of a simple video game developed using StyleBlit plugin. Source: *FIT CTU*

The plugin was studied on various animated objects as well as on sequences featuring a moving camera, both with preset trajectory and free-control of the camera. The results were well aesthetically evaluated and may be seen on attached figures.

One of the great advances of the plugin is its ability to seamlessly combine various materials, or source styles in this context, applied to one individual object. This was first studied on a simpler model of few materials (see figure 4.13) with a basic set of source textures.

The multi-material objects provide developers with innumerable possibilities to capture either traditional or their own art style which is unified in any way. Practical experiments were conducted on two scenes, as seen on figures 4.11, 4.14 and 4.15. These experiments show it is possible to create watercolor and pencil-drawn imagery using the plugin, with employable results required for video game development. On the other hand, those experiments revealed some limitations of technical and aesthetical character.



Figure 4.13: Multi-material testing on a single model. Model source: *3DMAesen*



Figure 4.14: A watercolor imitation on a simple scene. Model source: *Reallusion, Vertex Studio*

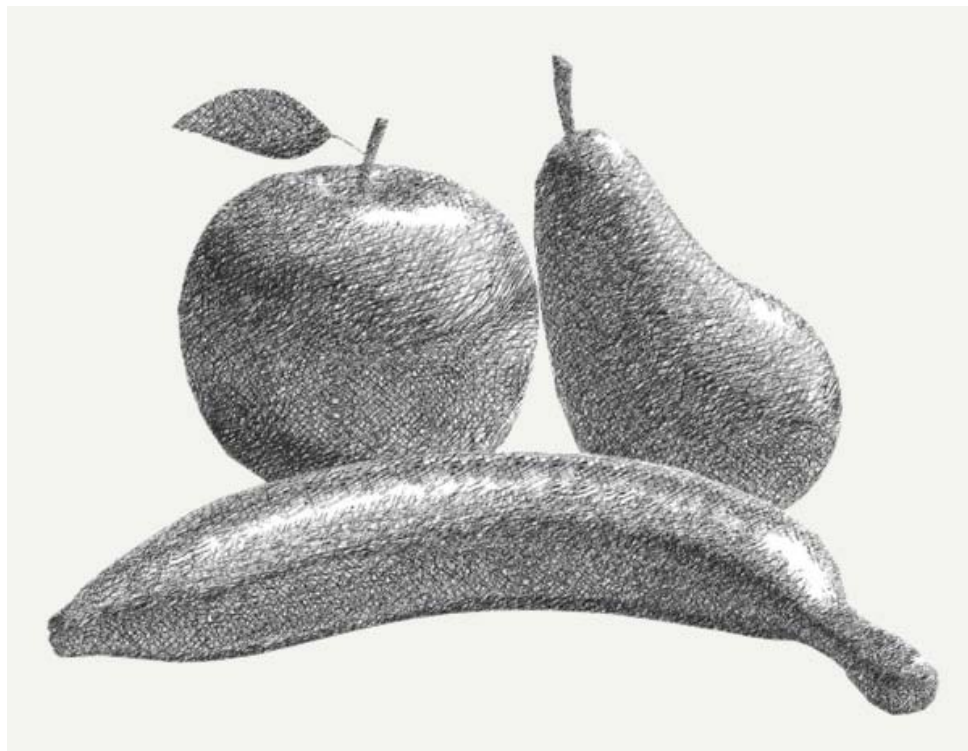


Figure 4.15: A pencil-drawn image imitation on a simple scene. Scene source: *astro3d*

4.5.4 Limitations and future work

As was mentioned numerous times, the inherent issue of the StyleBlit method concerns monotonous surfaces, i.e. surfaces with constant guidance values or little shift of those values: On any such surface, the source style texture is applied in a form of small equal patch(es), repeated over and over along the surface, creating pattern maps. Thus, use of StyleBlit on a surface such as the one on a figure 4.16 exhibits this phenomenon on each of the object's sides. Naturally, this behavior occurs in a detailed close-up view of virtually any object.

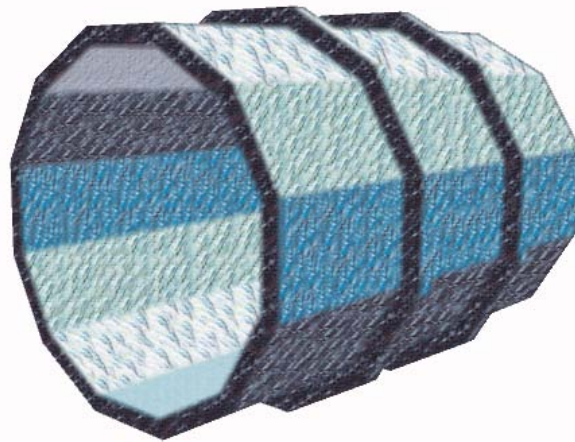


Figure 4.16: The body of a barrel exhibits unappealing repeated patterns because of the constitution of flat surfaces. Model source: *Wand And Circles*

It is also advisable to subject the model creation/selection to the art style of the project. Generally, well-rounded objects lacking hard edges are more fit to be stylized, as they are more likely to minimize the number of flat and monotonous surfaces, and at the same time complement most of the styles by nature. For instance, figures 4.14 and 4.11 both involve watercolor stylization, however, the man in an armchair seems more fitting for the style, blending different colors and variations, whereas the Japanese character seems too complex for the applied style. This was later confirmed with the pencil-drawn stylization (see figure 4.15), where the non-complex character of the fruit props is just fitting for the style exhibiting suppression of art complexity.

Of course, any limitation of the original StyleBlit algorithm is generally inherited by the Unity implementation, which includes non-enforcement of chunk coherency - textures featuring regular patterns are disjointed. This further demonstrates the notion the StyleBlit achieves the best result with stochastic textures. Other limitations include effectiveness decline if guidance

is strongly misaligned - e.g. if rotated upside down, in which case the method reduces the size of transferred texture chunks.

There are a few technical limitations of the plugin itself. One of them is the demand for limiting the angle of the camera or producing a workaround, as further explained in Unity setting and pitfalls section. Also, the practical use of the plugin is prone to corruption by inadequate settings by the developer, and the overall quality of the result is highly dependent on the settings of the video game. This reflects the fact that the method uses auxiliary textures, that need to be as precise as possible, e.g. the source guidance texture, which is used to produce the LUT. Violation of the highest quality input demand may cause fuzzy or blurred results, although, from the standpoint of Unity plugin creation, there is only so much that can be done to ensure the highest quality settings. Respecting the quality demands, especially the source style texture, is up to a developer.



Chapter 5

Conclusion

A plugin for Unity game engine, which implemented the StyleBlit algorithm, was successfully created. The plugin, provided with a texture of arbitrary style, is capable of producing complete stylized scenes. The plugin was tested during game development, collecting development inputs that were used to enhance the plugin. Performance test ran on a sample scene show it is possible to deploy the plugin in video game project development. A comparison with a vanilla version of StyleBlit implementation was evaluated with positive results. The plugin is capable of producing stylized scenes with objects consisting of multiple materials, combining different style textures.



Appendix A

Bibliography

- [Ami18] Amid Amidi. Aardman’s new video game 11-11: Memories retold shows a different side of the studio. *Cartoon Brew*, Sep 2018.
- [And14] Eric A. Anderson. On the rocks... <http://the-witness.net/news/2013/09/on-the-rocks>, Sep 2014.
- [Bla16] Peter Blaškovič. Rebelle: Real watercolor and acrylic painting software. In *ACM SIGGRAPH 2016 Appy Hour*, SIGGRAPH ’16, pages 3:1–3:2, New York, NY, USA, 2016. ACM.
- [Blo10] Jonathan Blow. Graphics tech: Precomputed lighting. <http://the-witness.net/news/2010/03/graphics-tech-precomputed-lighting>, Mar 2010.
- [Bro] Nikki Brown. Making the ultimate lux login – league of legends. <https://nexus.leagueoflegends.com/en-us/2016/12/making-the-ultimate-lux-login>.
- [Cap16] Edouard Caplain. The artist behind life is strange. *80 level*, Mar 2016.
- [DBB⁺17] Marek Dvorožňák, Pierre Bénard, Pascal Barla, Oliver Wang, and Daniel Sýkora. Example-based expressive animation of 2d rigid bodies. *ACM Transactions on Graphics*, 36, 07 2017.
- [Dec96] Philippe Decaudin. Cartoon-looking rendering of 3d-scenes. *Syntim Project Inria*, 6, 1996.
- [DLGKS18] Marek Dvorožňák, Wilmot Li, Vladimir G. Kim, and Daniel Sýkora. Toonsynth: Example-based synthesis of hand-colored

- cartoon animations. *ACM Transactions on Graphics*, 37:1–11, 07 2018.
- [Far16] Brian Fargo. Paper mario: Color splash dev on paint inspiration, approach to combat, says team put in its full effort. *Nintendo Everything*, Dec 2016.
- [FJL⁺16] Jakub Fišer, Ondřej Jamriška, Michal Lukáč, Eli Shechtman, Paul Asente, Jingwan Lu, and Daniel Sýkora. Stylit: illumination-guided example-based stylization of 3d renderings. *ACM Transactions on Graphics*, 35:1–11, 07 2016.
- [FJS⁺17] Jakub Fišer, Ondřej Jamriška, David Simons, Eli Shechtman, Jingwan Lu, Paul Asente, Michal Lukáč, and Daniel Sýkora. Example-based synthesis of stylized facial animations. *ACM Transactions on Graphics*, 36:1–11, 07 2017.
- [GDC16] GDC. Low complexity, high fidelity: The rendering of inside. <https://www.youtube.com/watch?v=RdN06E6Xn9E>, Dec 2016.
- [GGSC98] Amy Gooch, Bruce Gooch, Peter Shirley, and Elaine Cohen. A non-photorealistic lighting model for automatic technical illustration. In *SIGGRAPH*, volume 98, pages 447–452, 1998.
- [Hod19] Christopher Hodges. The 30 most expensive video games ever made. *TheGamer*, Mar 2019.
- [JvST⁺19] Ondřej Jamriška, Šárka Sochorová, Ondřej Texler, Michal Lukáč, Jakub Fišer, Jingwan Lu, Eli Shechtman, and Daniel Sýkora. Stylizing video by example. *ACM Transactions on Graphics*, 38(4), 2019.
- [Luq12] Raul Reyes Luque. The cel shading technique. Technical report, Citeseer, 2012.
- [Mam12] Jordan Mammo. The aesthetic failure of okami. *Unwinnable*, Jun 2012.
- [MFE07] Jason L Mitchell, Moby Francke, and Dhabih Eng. Illustrative rendering in team fortress 2. In *ACM SIGGRAPH 2007 courses*, pages 19–32. ACM, 2007.
- [Mon] Square Enix Montreal. Making lara croft go. *Unity Connect*.
- [MR05] Maic Masuch and Niklas Röber. Game graphics beyond realism: Then, now, and tomorrow. 01 2005.
- [MSB⁺17] Santiago E Montesdeoca, Hock Soon Seah, Pierre Bénard, Romain Vergne, Joëlle Thollot, Hans-Martin Rall, and Davide Benvenuti. Edge-and substrate-based effects for watercolor stylization. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, page 2. ACM, 2017.

- [MSR16] Santiago E Montesdeoca, Hock Soon Seah, and H-M Rall. Art-directed watercolor rendered animation. In *Proceedings of the Joint Symposium on Computational Aesthetics and Sketch Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering*, pages 51–58. Eurographics Association, 2016.
- [MSRB17] Santiago E Montesdeoca, Hock Soon Seah, H-M Rall, and Davide Benvenuti. Art-directed watercolor stylization of 3d animations in real-time. *Computers & Graphics*, 65:60–72, 2017.
- [MTD04] Xiao-Feng Mi, Min Tang, and Jin-Xiang Dong. Droplet: a virtual brush model to simulate chinese calligraphy and painting. *Journal of Computer Science and Technology*, 19(3):393, 2004.
- [RG18] Inc. Riot Games. Annie: Origins | behind the scenes | league of legends - youtube. <https://www.youtube.com/watch?v=v2n-1Cjwf6g>, Feb 2018.
- [Ric18] Madeline Ricchiuto. Go behind the scenes of ni no kuni ii: Graphics, lighting, and roland. *Bleeding Cool*, Mar 2018.
- [SJT⁺19] Daniel Sýkora, Ondřej Jamriška, Ondřej Texler, Jakub Fišer, Michal Lukáč, Jingwan Lu, and Eli Shechtman. StyleBlit: Fast example-based stylization with local guidance. *Computer Graphics Forum*, 38(2):83–91, 2019.
- [SS02] Thomas Strothotte and Stefan Schlechtweg. *Non-photorealistic Computer Graphics: Modeling, Rendering, and Animation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2002.
- [Ste] Eskil Steenberg. Love - online procedural adventiure game. <http://www.queilsolaar.com/love/development.html>.
- [Vog15] Mitch Vogel. Kirby and the rainbow curse developers give background on design decisions. *Nintendo Life*, Feb 2015.
- [WLS02] Der-Lor Way, Yu-Ru Lin, and Zen-Chung Shih. The synthesis of trees in chinese landscape painting using silhouette and texture strokes. *Journal of WSCG*, 10:499–506, 01 2002.

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Burýšek** Jméno: **Jiří** Osobní číslo: **398126**
Fakulta/ústav: **Fakulta elektrotechnická**
Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**
Studijní program: **Otevřená informatika**
Studijní obor: **Počítačová grafika a interakce**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Nefotorealistické zobrazování v herním engine s využitím výtvarné předlohy

Název diplomové práce anglicky:

Example-based Non-photorealistic Rendering using Game Engine

Pokyny pro vypracování:

Prostudujte algoritmy řešící problém přenosu stylu z výtvarné předlohy na složitější 3D modely [1, 2]. Implementujte algoritmus StyleBlit [2] v herním engine Unity úpravou jeho vykreslovacího řetězce prostřednictvím specializovaných shaderů a multipass renderingu. Vyhodnoťte výsledky implementace z hlediska kvality přenosu výtvarného stylu a z hlediska rychlosti zobrazování. Vyhodnoťte potenciál implementace pro uplatnění ve složitějším herním prostředí. Testování proveďte na nejméně dvou interaktivních scénách různé složitosti.

Seznam doporučené literatury:

- [1] Fišer et al.: StyLit: Illumination-Guided Example-Based Stylization of 3D Renderings, ACM Transactions on Graphics 35(4):92, 2016.
[2] Sýkora et al.: StyleBlit: Fast Example-Based Stylization with Local Guidance, submitted to SIGGRAPH 2018.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Daniel Sýkora, Ph.D., Katedra počítačové grafiky a interakce

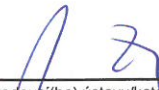
Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **15.02.2018**

Termín odevzdání diplomové práce: **24.05.2019**

Platnost zadání diplomové práce: **30.09.2019**


doc. Ing. Daniel Sýkora, Ph.D.
podpis vedoucí(ho) práce

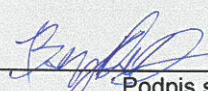

podpis vedoucí(ho) ústavu/katedry


prof. Ing. Pavel Ripka, CSc.
podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

20.12.2018
Datum převzetí zadání


Podpis studenta