

“ALEXANDRU IOAN CUZA” UNIVERSITY OF IAŞI
FACULTY OF COMPUTER SCIENCE



BACHELOR THESIS
AI GAME ENGINE

proposed by

Şandru Maria
Session: July, 2021

Scientific Coordinator:
Conf. dr. Mădălina Răschip

“ALEXANDRU IOAN CUZA” UNIVERSITY OF IAŞI
FACULTY OF COMPUTER SCIENCE

AI GAME ENGINE

Şandru Maria

Session: July, 2021

Scientific Coordinator:
Conf. dr. Mădălina Răschip

Anexa 4

Avizat,

Îndrumător Lucrare de Licență

Titlul, Numele și prenumele _____

Data _____ Semnătura _____

DECLARAȚIE privind originalitatea conținutului lucrării de licență

Subsemnatul(a) SANDRU MARIA
domiciliul în AOROCOI
născut(ă) la data de 27.01.2000, identificat prin CNP 6000127071382,
absolvent(a) al(a) Universității „Alexandru Ioan Cuza” din Iași, Facultatea de
INFORMATICA specializarea ENGLEZĂ, promoția 2021, declar
pe propria răspundere, cunoscând consecințele falsului în declarații în sensul art. 326
din Noul Cod Penal și dispozițiile Legii Educației Naționale nr. 1/2011 art.143 al. 4 și 5
referitoare la plagiat, că lucrarea de licență cu titlul:
AI GAME ENGINE

MĂDALINA RASCHIȚ elaborată sub îndrumarea dl. / d-na
pe care urmează să o susțin în fața
comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată
prin orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la
introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări
științifice în vederea facilitării fasificării de către cumpărător a calității de autor al unei
lucrări de licență, de diplomă sau de disertație și în acest sens, declar pe proprie
răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am
întreprins-o.

Date azi, 23.06.2021

Semnătură student



Anexa 5

DECLARAȚIE DE CONSUMĂMÂNT

Prin prezența declar că sunt de acord ca Lucrarea de licență cu titlul „AI GAME ENGINE”, codul sursă al programelor și celealte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribue în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, data

23.06.2021

Absolvent *Prenume Nume (în clar)*

MARIA SANDRU

(semnătura în original)



ACORD PRIVIND PROPRIETATEA DREPTULUI DE AUTOR

Facultatea de Informatică este de acord ca drepturile de autor asupra programelor-calculator, în format executabil și sursă, să aparțină autorului prezentei lucrări, Prenume Nume.

Încheierea acestui acord este necesară din următoarele motive:

[Se explică de către autor de ce este necesar un acord, se descriu originile resurselor utilizate în realizarea produsului-program (personal, tehnologii, fonduri) și aportul adus de fiecare resursă.]

Lucrarea folosește API-ul GPT-3 de la OpenAI și este interfață prin contract cu compania GOOGLE dreptul altei persoane de a folosi acest API care nu este public
Iași, data 23.08.2021

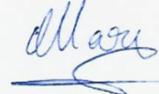
Decan Prenume Nume

(semnătura în original)

Absolvent Prenume Nume

MARIA SANDRU

(semnătura în original)



Introduction	8
Motivation	11
Contributions	13
The description of the application	14
The addressed problem	14
The proposed solution	15
Application's functionalities	16
Similar solutions	16
Unreal Engine	16
Unity	17
AI Dungeon	18
Façade	19
Vade Mecum	19
Application's architecture	21
Introduction	21
Responsiveness	21
Rendering Pipeline	21
The Physically Based Rendering pipeline module	23
Surface reflection	26
Normal distribution function	28
Geometry function	29
Fresnel equation	30
The reflectance equation	32
The microfacet surface model	32
Material parameters	33
Base Color	33
Normal	34
Metallic	34
Roughness	34
Comparisons between rendering in Yūgen Engine and Blender	36
Natural Language Processing Pipeline	39
APIs	42
Case studies	44
Generating stories	44
Creating photorealistic environments and characters	48

Conclusions	51
Future Improvements	52
Bibliography	53

Introduction

How many times has one read the final page of a book and hoped the storyline would continue? The story did not just entertain; it moved the audience. It was so poetic, challenging, and metaphorically resonant that one wholeheartedly wished to be able to keep living in that world?

A melancholy awareness that the universe described in those pages will fade away from one's memory takes over. An acute sensitivity to the transience of lovely things combined with a rich enjoyment of this short-lived beauty provokes an emotion like nothing else as one closes the book. A poignant feeling of ephemerality, a beautiful sadness, affects the readers immediately and involuntarily before putting that feeling into words.

One might get lucky if the series continues or the author releases a sequel for the most prominent franchises, but for the vast majority of worlds, one will never get to experience it again. Even so, one can not help but dream of feeling once again the heightened state of emotion created by that enthralling piece of literature.

A possible soul-soothing consolation would be that this sensation can not only train the public to become more sensitive to beauty, but it can help them reach more of an acceptance of the mortality and transience inherent to life. The perishable aspect of things, which is a tremendous cause of grief, gives them such distinctive beauty and emotional force.

If adventurous enough in daydreaming, one can imagine a world where one can build a fictional reality in an unlimited number of different ways, where each experience, character, and location is unique and alive, where worlds alter as a result of one's choices, and where the people who live in those worlds feel energetic and genuine. That story could be the dearest novella, movie, or even an utterly novel narrative shaped entirely by the lecturers, where they are fully immersed and can experience indefinitely.

The lovely tale can continue in unexpected ways wholly unforeseen by the original creator. A reader could dine with a beautiful ghost, be the savior the people of a cursed land pray for or be awakened by one of the servants alarmed that the keep is under attack.

The good news is that this fantasy is not an unattainable dream, courtesy of the Yūgen Engine, which is a game engine empowered by artificial intelligence that allows the complete immersion into one's ideal fictional worlds that take the form of video games. The difference from traditional game engine productions is that the developer's limited options do not confine the players. The non-playing characters are not merely art assets with some handcrafted dialog, and worlds are not frozen walled-off boxes.

The motivation behind Yūgen Engine is the insight that because of the improved efficacy of natural-language creation in artificial intelligence, there are now a growing number of examples of interactive fiction that employs AI to dynamically generate fresh, open-ended content rather than relying on pre-written material.

The vision of the Yūgen Engine was led by realizing that no game engines are available for developing games that provide the same freedom to do anything as in DnD or AI Dungeon. This game engine allows the production of games where one has complete control over one's actions by utilizing strong natural language generation models such as GPT-3 to determine what happens in a story.

The thesis is structured in four main chapters that will help readers understand the application structure, present the details related to the implementation, and introduce the technologies used.

Chapter 1 - The Description Of The Application provides a glimpse into what the Yūgen Engine can do and the current technological context that focuses on game characters embarking upon imaginary adventures within an open-world fantasy setting with the grace of artificial intelligence. This chapter serves as a gentle introduction to the revolutionizing aspect of the engine: the existence of a medium dedicated to producing cutting-edge graphics where the game narratives result from natural language processing techniques. This section illustrates how the Yūgen Engine accomplishes the ambition of building a multiverse of countless worlds powered by AI that allows players to continue to explore and experience the worlds they love. Also, the highlight is on the functionalities: the customization of 3D worlds and the creation of NLP-powered stories and dialogues. Moreover, here are described previous inspirations that opened the door for imagining such an engine.

Chapter 2 - The Application's Architecture gives a detailed account of the technical details of the game engine's architectural approach based on theoretical notions. It presents how the application's features were implemented and integrated with a plethora of APIs and the reasoning behind those choices.

Chapter 3 - The Use Cases paints in words two scenarios: rendering a photorealistic 3D environment using the Yūgen Engine's UI and making an infinitely generated text adventure powered by deep learning with tales and dialogues.

Chapter 4 - Conclusions gives details of the final thoughts on the application's current state and future potential. It discusses possible features that would further the engine's innovative nature—for example, giving NPCs personalities and moral development or bettering the NLP-created tales with methods such as using generative adversarial neural networks.

As a final note, the name for this game engine feeds into the idea that the developers can create their creative exquisite oases using it. Yūgen is an untranslatable Japanese word that gives a name to a mood in which one feels that the whole universe possesses a mysterious, elusive, but natural beauty.

Motivation

We are inveterate cineastes in our minds. We imagine little movies in which we say what we mean, seize advantages that elude us in real life, and replay scenarios that might have ended differently. The longing to edit and manipulate reality could be satisfied with the aid of video games empowered by AI.

Video games are most and foremost a form of art that can console and educate our souls while giving the thrill of escapism from the sorrows of daily life. Therefore, they deserve to be honored as a noble creative endeavor, a few hours of highly complex mental choreography from the player. In its best instances, they draw upon the same faculties as those who underpin great art, the pinnacle of human achievement and meaning.

Digital games are akin to the art one sees in a gallery or museum. Some day, one might want to gaze at a captivating landscape whose equivalent in gaming might be something like “Firewatch.” Another day, one could want to stand before Picasso’s “Guernica” and feel the pain and misery of the Spanish Civil War, an experience closer to something like playing “This War of Mine” about a group of civilians trying to survive in a besieged city.

Even opera could be a good analogy for video games, particularly in terms of the concept of the total work of art. Opera works combine music, drama, dance, costumes, set design, and everything else into one harmonious whole, rather than all the skills existing separately. Similarly, video games are compilations of various people’s creative output: sound engineers, writers, designers, programmers, composers, concept artists, modelers, directors, and many other people.

Due to its interactive nature, sculpting also has a strong resemblance to video games. To fully appreciate sculptures and installations, the admirers must walk around them. More and more works of art assume and require viewer participation, but few do so naturally as a video game. Instead, they respond to players and provide a variety of experiences based on their choices. Because of the high level of interactivity, the player becomes, at least in part, a co-author.

Gamers can try to figure out what the developer is trying to say or achieve or use the experience to embody or project their worldview.

Video games have something from traditional artistic ventures such as painting, performing in an opera, or sculpting while bringing dignity to the imagination and a sense of freedom. Considering the argument that video games are indeed a mentally rewarding activity, the game developer's goal should be to make the gaming experience as engaging and immersive as possible. The potential of artificial intelligence in game production is immense, considering the impact that games like Façade have had. These games revolutionized the way players express themselves and are involved in the game narratives because they could be understood without any pre-imposed phrases, which felt like an unprecedented dose of freedom. Therefore, the Yūgen Engine was designed in the hopes of developing NLP-empowered video games quickly and efficiently.

Contributions

The Yūgen Engine is a unique 3D game engine that allows developers to build infinite AI-powered scenarios that can change the way we experience visionary universes. These created worlds are continuously evolving, rewarding users for their imagination and creativity.

The game creators can design photorealistic or stylized environments and ingenious characters. The latter are empowered with a conscience of their own, expressing thoughts, memories, and feelings without the help of any pre-established phrases. Instead, they give gamers beautifully expressed executive summaries of what they feel at a given point laid out in generous and imaginative manners, like something a great novelist would invent.

The description of the application

The addressed problem

Video games are fantastic at telling stories and letting the plot unfold over time. They are paced like an enchanting novel or movie, with fast-paced action interspersed with slower exposition and character development moments. They allow one to build ever-evolving worlds and explore the amazingly intricate worlds that others have created.

However, as open-world and narrative-based games become more sophisticated and newer PCs and consoles display increasingly realistic and detailed landscapes, more advanced deep learning mechanisms are needed. Furthermore, it will be strange and alienating to be pushed into a nearly photorealistic world full of elaborate systems and story possibilities, only to realize that non-player characters still behave like soulless machines as most NPCs function in AAA game productions. Nonetheless, the notion that gamers can only express and act by having at their disposal a few options given by the game creator strips away the individual's liberty and originality. Last but not least, another daunting aspect of modern games is that there is a fixed storyline and world.

The next generation of open-world adventures will respond to the players with missions tailored to the gamers' preferences and activities and characters who can imitate empathy and aggression. NLP in games would allow AIs to construct human-like conversational pieces and then genuinely speak them, eliminating the requirement for pre-recorded lines of speech spoken by an actor. Gamers could have people in the game world retelling stories about the things the players have done. Rumors and gossip will be exchanged between NPCs, as will myths and legends. Take, for example, arriving in a village in The Witcher to find a minstrel singing songs about the gamer's last dragon encounter or the particular way the players have dealt with the Bloody Baron.

The AI researcher Julian Togelius said in 2020, "I fully expect that within a year, someone else will have essentially implemented GPT-3 in a game." in order to signal how NLP could be the future of video games.

The proposed solution

Over the last decade, artificial intelligence has progressed at an unprecedented rate. With the release of AI Dungeon, it has been demonstrated that a natural language AI can be used to create enjoyable and engaging experiences for the first time. NLP provides players with something no other game has ever done before: complete freedom, an opportunity for self-expression and imagination, and limitless options.

The next step would be a game engine capable of creating games like AI Dungeon or Facade, which is the purpose of this game engine. Yūgen includes rich worlds, each with its kingdoms and towns, factions, classes, and characters. The universes are original and varied, filled with infinite detail. Anything novel, whether made by people, as a feature film or made by pure fantasy such as French-speaking unicorns, is a recombination of previously existing features in a new way. The characters are also charming agents with their own emotions. They are equipped with the function of being thoughtful companions, having fruitful and enriching perspectives on life, being curious about one's inner world while at the same time generously offering their thoughts.

The Yūgen Engine can be thought of as the Wizard of Oz, only that AI is the magic behind the curtain. We just might be shocked by what we find when we pull back the curtain.

Application's functionalities

The Yūgen Engine accomplishes the following: the creation of photorealistic and stylised worlds and the generation of interactive, unique stories and dialogues.

Similar solutions

Unreal Engine

Unreal Engine¹ is the world's most open and advanced real-time 3D creation platform for photoreal visuals and immersive experiences. It is a complete suite of creation tools for game development, architectural and automotive visualization, linear film and television content creation, broadcast and live event production, training and simulation, and other real-time applications (Figure 1).

Whether one is interested in more convincing animated characters, enticing natural worlds and environments, comprehensive virtual production toolset, enhanced high-quality media output, Unreal proves to be the state-of-the-art tool.

¹ <https://www.unrealengine.com/en-US/>



Figure 1: Unreal Engine

Unity

Unity² is a robust cross-platform game development environment and runtime engine supporting a wide range of platforms. Unity's primary design goals are ease of development and cross-platform game deployment (Figure 2). As such, Unity provides an easy-to-use integrated editor environment, in which one can create and manipulate the assets and entities that constitute one's game world, which served as inspiration for The Yūgen Engine. Unity also provides a robust suite of tools for analyzing and optimizing one's game on each target platform, a comprehensive asset conditioning pipeline, and the ability to manage the performance-quality trade-off uniquely on each deployment platform. In addition, Unity supports scripting in JavaScript, C# or Boo; a powerful animation system supporting animation retargeting (the ability to play an animation authored for one character on a different character); and support for networked multiplayer games.

² <https://unity.com/>

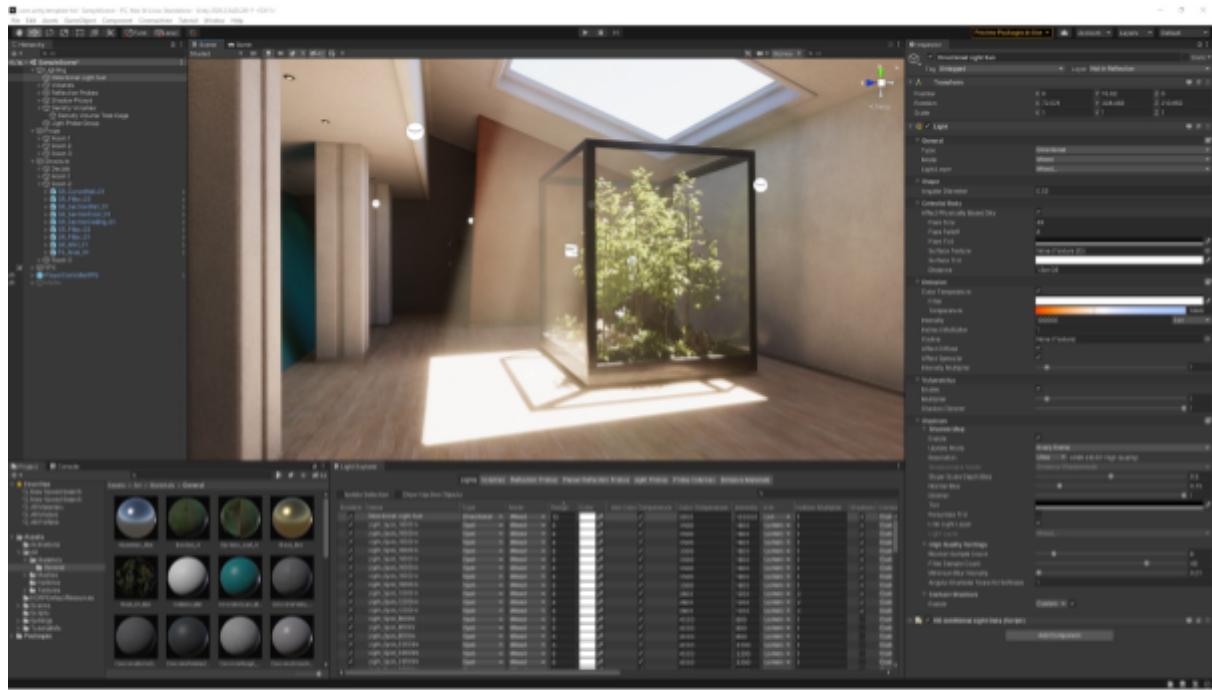


Figure 2: Unity

AI Dungeon

AI Dungeon³ is a free-to-play single-player and multiplayer text adventure game that uses artificial intelligence to generate content. It also enables players to create and share their custom adventure settings. Their vision is to build a multiverse of countless worlds powered by artificial intelligence that allows users to continue to explore and experience the worlds they love.

The fact that AI has superpowers is enhanced in AI Dungeon, revolutionizing the way we play. One can train AI game systems on new content types and completely change the genre, style, or world they give to someone, unlike traditional game systems. As a result, dreamers will be able to continue to experience and live out the stories of their favorite worlds for the first time.

The Yūgen Engine took AI Dungeon as a source of inspiration mimicking the process of generating content.

³ <https://play.aidungeon.io/main/landing>

Façade

Façade⁴ is a robust interactive fiction experience that incorporates elements of both interactivity and drama, uses voice acting, 3D environments and characters, natural language processing, and other advanced artificial intelligence routines. As in an interactive stage play, the player can take an active role in the conversation, pushing the topic in one direction or another without having pre-imposed phrases. These stage plays are saved as script text files, which the player can read after they have finished.

The Yūgen Engine's primary purpose is to generate games like Façade with hyper-realistic environments and NLP-generated interactions between non-playing characters and gamers, while the latter can express their thoughts with no confinement and have the storyline automatically generated.

Vade Mecum

Vade Mecum is a mystery game in which one is supposed to ask Elizabeth, a detective's assistant, questions about a criminal case to find the clues and to come up with the right story at the end. She is empowered by GPT-3 and Bert.

For the most part, the bot is there to validate one's assumptions. The bot can answer any question no matter how one puts it and is intelligent enough to understand any nuances one's final story has.

One can express oneself freely, and Elizabeth will understand, which is not common in video games where one usually has predefined lines to say or respond to. She can also hold meaningful conversations on various topics not related to the case if asked to (Figure 3 shows a conversation with Elizabeth).

This project was done during the "Techniques of natural language processing" university laboratory in collaboration withCarla-Diana Petrut. The Yūgen Engine is capable of creating games as such.

⁴ <https://www.playblstudios.com/facade>

VADE MECUM

A woman buys a pair of red high heels. Hours later she dies.
Ask Elizabeth (your assistant) questions in order to solve the mystery.

Elizabeth:

Yes, I do. I love going to the beach, hiking in the mountains, and exploring new places. It's a great way to unwind after a long day at work. What do you like to do when you're not working? What are some of your favorite places to go o

Do you enjoy spending time outside?

Send

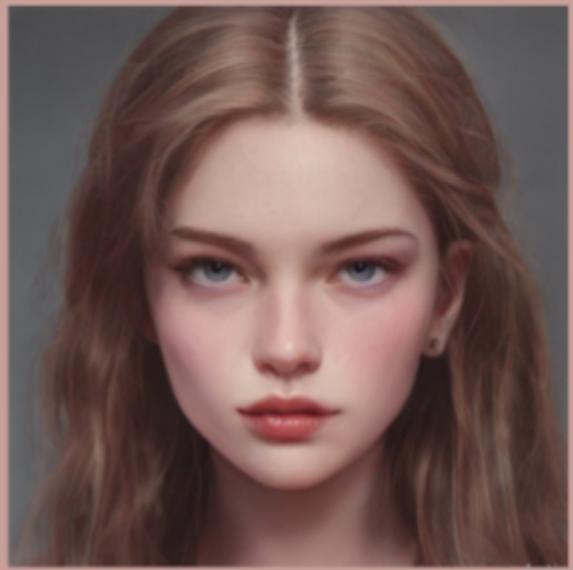


Illustration by Chantal D'Amico

Figure 3: VADE MECUM

Application's architecture

Introduction

Yūgen contains a data-driven architecture, and this is what differentiates it from a piece of software that is a game but not an engine. For example, when a game contains hard-coded logic or game rules or employs special-case code to render specific game objects, it becomes difficult or impossible to reuse the software to make a different game. Instead, Yūgen is a tool that is extensible and can be used as the foundation for many other games without significant modification.

The Yūgen Engine is configured as a dynamic-link library. One of the reasons would be not to have it copied into the executable at compile time but rather compile and link the code into a loadable form by programs/ games at run time and link time.

Responsiveness

This component includes the messaging and event system and the layers which form the basis of any game engine.

Rendering Pipeline

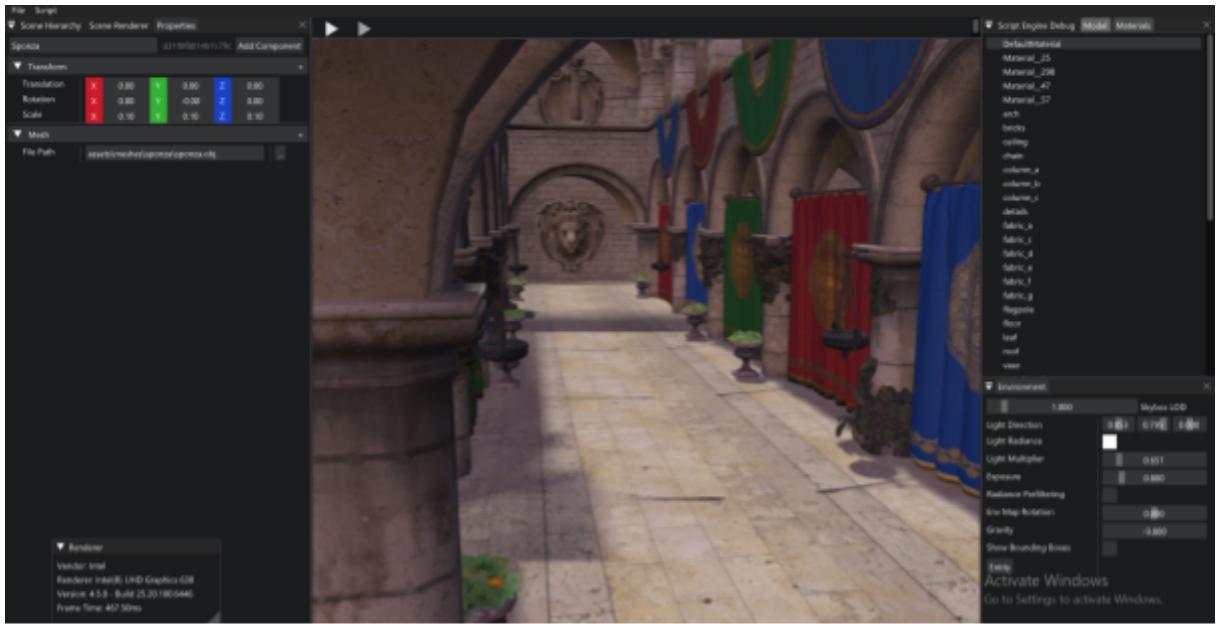


Figure 4: Scene rendered in Yūgen Engine

The rendering of computer graphics is a vital component. At the highest level of abstraction, the rendering process transforms a three-dimensional scene description into a picture. In addition, animation algorithms, geometric modeling, texturing, and other computer graphical domains require a rendering technique to make their findings appear in an image. As a result, rendering has grown ubiquitous, opening new vistas for creative expression, entertainment, and visualization in everything from movies to games and beyond (Figure 4 and Figure 5 offer a glimpse of a scene rendered in Yūgen).

Yūgen and the contents of this paper concentrate entirely on photorealistic rendering, which is the goal of creating images that cannot differ from the photos captured by a camera or the process of creating images that elicit the same emotion as gazing at the natural environment. There is a variety of reasons to concentrate on photorealism:

- Computer-generated imagery must be combined smoothly with real-world film; photorealistic visuals are critical for the movie special-effects business.
- Photorealism is a powerful strategy for helping the spectators forget that they are gazing at an environment that does not exist in entertainment applications when all pictures are synthetic.
- Photorealism provides a well-defined criterion for assessing the output quality of a rendering system.

Almost every photorealistic rendering method is based on the technique for ray tracing. Ray tracing is a reasonably simple method; it is based on trailing a light ray through a scene when interacting with objects and rebounding them off the surroundings. Although a ray tracer may be written in numerous ways, the Yūgen Engine imitates the following entities and phenomena:

Cameras: A camera model defines how and from where a scene is observed and how the scene's picture is captured on a sensor. Many rendering techniques start with the camera and trace viewing rays into the scene.

Intersections of rays and geometric objects: We need to know precisely where a ray meets a geometric item. In addition, some features of the object at the crossing point, such as a surface normal or its substance, must be determined. Therefore, most ray tracers also include the ability to test the intersection of a ray with numerous objects, returning the intersection that is closest to the ray's origin.

Light sources: Rendering a scene would be pointless if it didn't have lighting. A ray tracer must model the dispersion of light throughout the scene, not just the positions of the lights but also how they spread their energy over space.

Visibility: To determine if a particular light deposits energy at a point on a surface, we must first determine if the path from the point to the light source is continuous. Fortunately, this issue is simple to answer with a ray tracer: we just create the ray from the surface to the light, locate the nearest ray-object junction, and compare the intersection distance to the light distance.

Surface scattering: Each item must describe its appearance, including how light interacts with the item's surface and the type of the reradiated (or dispersed) light. Surface scattering models are often parameterized to replicate a wide range of appearances.

Indirect light transport: Because light might arrive at a surface after bouncing off or traveling through other surfaces, tracing extra rays originating at the surface is frequently required to completely depict this impact.

Ray propagation: We need to understand what happens to light as it travels along a ray across space. Light energy remains constant along a ray while rendering a scene in a vacuum. True vacuums are uncommon on Earth, although they are an acceptable approximation in many situations. More advanced models are available to trace rays through the fog, smog, the Earth's atmosphere, and other obstacles.

The Physically Based Rendering pipeline module



Figure 5: Scene rendered in Yūgen Engine

Yūgen Engine's ambition is to approximate reality, and, for honoring this goal, it uses the physically-based-rendering approach. PBR[1] is a computer graphics approach that seeks to render images that model the flow of light in the real world. Compared to other lighting algorithms like Phong and Blinn-Phong, physically-based rendering looks more realistic because it aims to mimic light in a physically plausible way. Not only does it look better because it accurately represents actual physics, but it also allows game developers and artists to author surface materials based on physical parameters rather than relying on tweaks to get the lighting to look photorealistic.



Figure 6: Scene rendered in Yūgen Engine

PBR is a ray-tracing-based workflow that is generally acknowledged in the CG industry and is utilized in most leading industry 3D software and gaming engines, including Substance Painter, 3ds Max, Maya, Blender, Unreal Engine, and Unity. Also, physically-based-rendering is used widely for producing computer-generated imagery for movies such as "The Hobbit: The Desolation of Smaug" (2013) and "The Hobbit: The Desolation of Smaug" (2014). Physically-based Monte Carlo techniques to rendering have been successful in production for several reasons, one of which is that they increase artist productivity. [1] Some of the essential factors have been:

- Surface material design has become more accessible as a result of the adoption of physically-based reflection models. Previously, when non-energy-saving reflection models were utilized, an object could be placed in a single lighting environment while its surface reflection characteristics were modified. As a result, the asset would look delightful in one setting. Still, it would often appear entirely flawed when transported to another lighting situation because the surfaces reflected too little or too much energy: surface characteristics had been adjusted to absurd values. Figure 6 and Figure 7 show how physically based materials will work equally well in all lighting environments by rendering a camera in the Yūgen Engine.

- Shadows computed with ray tracing are substantially better than those calculated with rasterization. As a result, lighting artists no longer have to fiddle with shadow map resolutions, biases, and other settings, which is a tedious operation. Furthermore, rather than being an artistically calibrated manual process, physically-based solutions include bounce lighting and other soft-lighting effects as part of the method.
- Ray-tracing methods are also suitable for both progressive refining and rough computing previews quickly using only a few samples per pixel; rasterization-based renderers lack these features.



Figure 7: Scene rendered in Yūgen Engine

Yūgen Engine is composed of the following concepts to use realistic shading/lighting models along with measured surface values to accurately represent real-world materials:

Surface reflection

When light strikes a surface, the light scatters, and some of it is reflected back into the environment. Two key impacts must be described to model this reflection: the reflected light's spectral distribution and its directional distribution. The skin of a lemon, for example, absorbs most light in blue wavelengths but reflects most light in red and green wavelengths. As a

result, when lit with white light, it turns yellow. As an observation, the color of the lemon peel is nearly identical no matter which direction it is viewed from, while a highlight, a brighter patch that is more white than yellow, is evident from some perspectives. The light reflected from a point in a mirror, on the other hand, is almost entirely dependent on the viewing direction.

A formalism for characterizing reflection from a surface is the bidirectional reflectance distribution function . BRDF is a method that takes the incoming light direction ω_i , the outgoing direction ω_o , the surface normal n , and a roughness value a for the microsurfaces. Given an opaque surface's material qualities, the BRDF estimates how much incident light along ω_i is scattered from the surface in the direction ω_o . For example, if the surface is smooth, as in a mirror, the BRDF function $fr(\omega_i, \omega_o, n, a)$ will return 0.0 for all incoming light rays ω_i except the one that has the same reflected angle as the outgoing ray ω_o , in which case the function will return 1.0. [1]

Physically-based BRDFs have two crucial qualities: reciprocity and energy conservation. For all pairs of directions, ω_i and ω_o , $fr(\omega_i, \omega_o, n, a) = fr(\omega_o, \omega_i, n, a)$ and the total energy of light reflected is less than or equal to the energy of incident light.

There are several physically based BRDFs out there to approximate the surface's reaction to light. However, almost all real-time PBR render pipelines use a BRDF known as the Cook-Torrance BRDF.

The Cook-Torrance BRDF contains both a diffuse and specular part:

$$f_r = k_d f_{lambert} + k_s f_{cook-torrance}$$

where k_d is the ratio of incoming light energy that gets refracted, with k_s being the reflected ratio. [2]

The left side of the BRDF states the diffuse part of the equation denoted here as $f_{lambert}$. This is known as Lambertian diffuse similar to what is used for diffuse shading, which is a constant factor represented as:

$$f_{lambert} = \frac{c}{\pi}$$

, with c being the albedo or surface color, and then division by π is there to normalize the diffuse light. [2]

The general Cook-Torrance microfacet specular shading model is:

$$f_{cook-torrance}(l, v) = \frac{D(h)F(v, h)G(l, v, h)}{4(n \cdot l)(n \cdot v)}$$

which uses the normal distribution function $D(h)$, the Fresnel equation $F(v, h)$, and the Geometry function $G(l, v, h)$, where the vector h is the halfway vector that sits halfway between the light l and view v vector, and n is the surface's normal. [2]

Normal distribution function

The relative surface area of microfacets perfectly aligned to the midway vector h is statistically approximated by the normal distribution function D . The Trowbridge-Reitz GGX is one of many NDFs (normal distribution functions) that statistically approximate the general alignment of microfacets given a roughness parameter:

$$D(h) = \frac{\alpha^2}{\pi((n \cdot h)^2 \cdot (\alpha^2 - 1) + 1)^2}$$

where h is the halfway vector for measuring against the surface's microfacets, with α denoting the roughness of the surface, many microfacets are aligned to midway vectors across a limited radius when the roughness is low, and therefore the surface is smooth. [2]

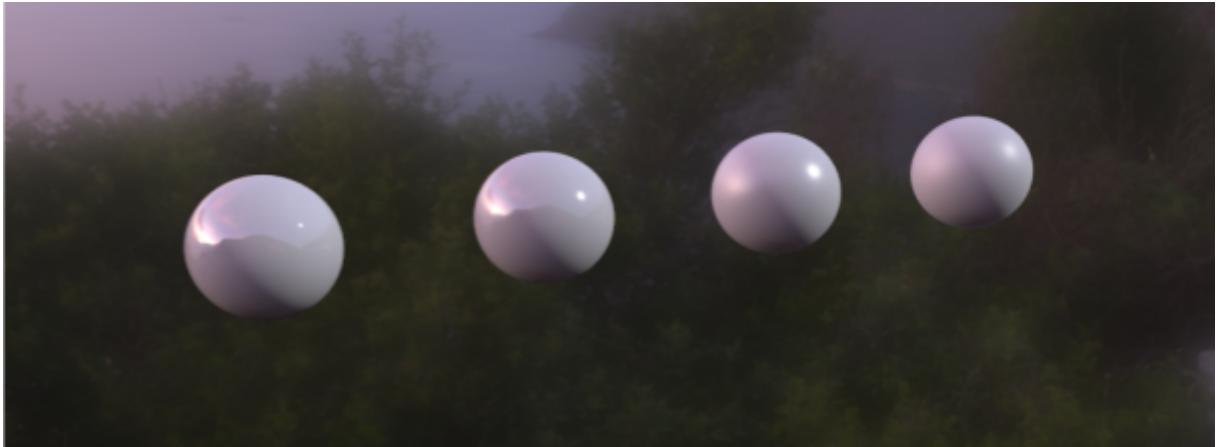


Figure 8: Scene rendered in the Yūgen Engine.

When the surface roughness is low, many microfacets are aligned to halfway vectors across a tiny radius, resulting in a smooth surface. Due to this high concentration, the NDF displays a very bright spot. However, on a rough surface, where the microfacets are aligned in much more random directions, one will discover a much higher number of midway vectors that are partially aligned to the microfacets but less concentrated (Figure 8).

In Yūgen Engine, the Trowbridge-Reitz GGX normal distribution function translates to the following code:

```
float NormalDistributionFunctionGGX(float n_dot_h, float roughness)
{
    float alpha = roughness * roughness;
    float alpha_squared = alpha * alpha;

    float denom = (n_dot_h * n_dot_h) * (alpha_squared - 1.0) + 1.0;
    return alpha_squared / (PI * denom * denom);
}
```

Geometry function

The geometry function calculates a statistical estimate of the percentage surface area where its microsurface features overlap, obstructing light beams. The Geometry function, like the normal distribution function, takes the roughness parameter of a material as an input, with rougher surfaces having a greater chance of overshadowing microfacets. The geometry function Yūgen Engine uses is Schlick-GGX, which is a mix of the GGX and the Schlick-Beckmann approximation:

$$G_{SchlickGGX}(v) = \frac{n \cdot v}{(n \cdot v)(l-k)+k}$$

where k is a remapping of α based on whether geometry function is used for either direct lighting or IBL lighting:

$$k_{direct} = \frac{(Roughness+1)^2}{8}$$

$$k_{IBL} = \frac{\alpha^2}{2}$$

We must account for both the view direction (geometry obstruction) and the light direction vector to estimate geometry (geometry shadowing). Using Smith's approach, we can account for both:

$$G(n, v, l, k) = G(n, v, k)G(n, l, k)[2]$$

The implementation is given next:

```
float GeometrySchlickGGX(float n_dot_v, float roughness)
{
    float r = (roughness + 1.0);
    float k = (r * r) / 8.0;

    float nom = n_dot_v;
    float denom = n_dot_v * (1.0 - k) + k;

    return nom / denom;
}

float GeometrySmith(vec3 n, vec3 v, vec3 l, float roughness)
{
    float n_dot_v = max(dot(n, v), 0.0);
    float n_dot_l = max(dot(n, l), 0.0);
    float ggx_2 = GeometrySchlickGGX(n_dot_v, roughness);
    float ggx_1 = GeometrySchlickGGX(n_dot_l, roughness);

    return ggx_1 * ggx_2;
}
```

Fresnel equation

The term Fresnel is used in computer graphics to describe the varying reflectivity that occurs at different angles. The light that strikes a surface at a grazing angle, for example, is considerably more likely to reflect than light that hits it directly. Consequently, reflections near the borders of objects displayed with a suitable Fresnel effect will appear brighter.

Once rendered, the Fresnel effect will add reflectivity to the artist-specified values, reaching completely white at glancing angles. The content essentially outlines the foundation, and Fresnel's equations then take over, making the surface more reflective at various angles as required (Figure 9).

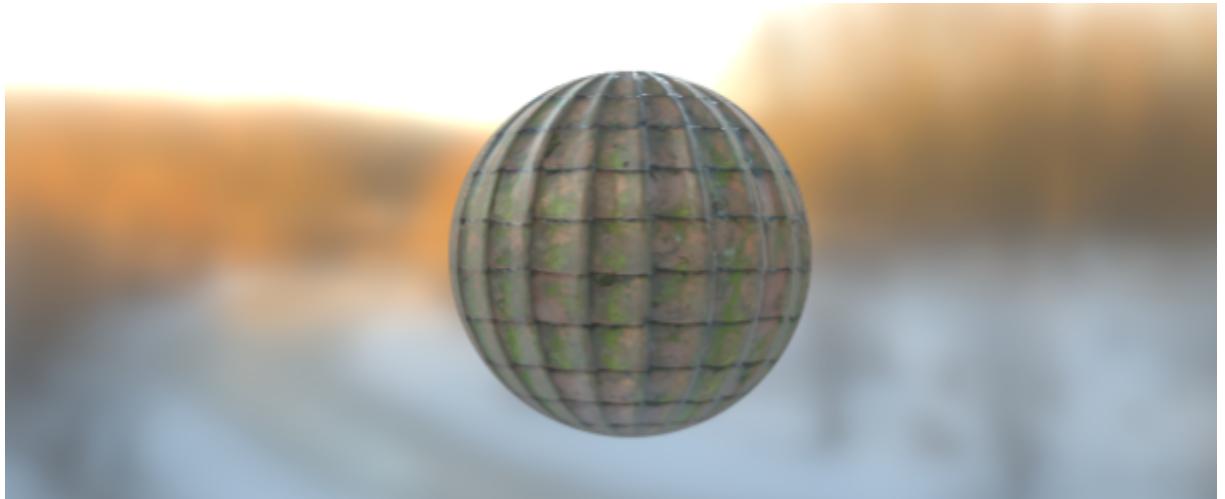


Figure 9: Scene rendered in Yūgen Engine

This phenomenon is described by the Fresnel equation:

$$F(v, h) = F_0 + (1 - F_0)2^{(-5.55473(v \cdot h) - 6.98316)(v \cdot h)}$$

, but because it is rather complex, it can be approximated as:

$$F_{Schlick}(h, v, h_0) = F_0 + (1 - F_0)(1 - (h \cdot v))^5.[2]$$

The base reflectivity of the surface or the minimum amount and color of light reflected is represented by F_0 , which is calculated using indices of refraction.

Some of the more common values for surface's response at normal incidence can be taken from Naty Hoffman's course notes listed below [3]:

Material	F_0 (Linear)	F_0 (sRGB)	Color
Water	(0.02, 0.02, 0.02)	(0.15, 0.15, 0.15)	
Plastic / Glass (Low)	(0.03, 0.03, 0.03)	(0.21, 0.21, 0.21)	
Plastic High	(0.05, 0.05, 0.05)	(0.24, 0.24, 0.24)	
Glass (high) / Ruby	(0.08, 0.08, 0.08)	(0.31, 0.31, 0.31)	
Diamond	(0.17, 0.17, 0.17)	(0.45, 0.45, 0.45)	

Iron	(0.56, 0.57, 0.58)	(0.77, 0.78, 0.78)	
Copper	(0.95, 0.64, 0.54)	(0.98, 0.82, 0.76)	
Gold	(1.00, 0.71, 0.29)	(1.00, 0.86, 0.57)	
Aluminium	(0.91, 0.92, 0.92)	(0.96, 0.96, 0.97)	
Silver	(0.95, 0.93, 0.88)	(0.98, 0.97, 0.95)	

In Yūgen Engine, the Fresnel Schlick approximation translates to:

```
vec3 fresnelSchlick(vec3 F0, float cosTheta)
{
    return F0 + (1.0 - F0) * pow(1.0 - cosTheta, 5.0);
}
```

The reflectance equation

This is a fundamental equation in rendering; it models how an incident distribution of light at a point is transformed into an outgoing distribution, based on the scattering properties of the surface:

$$L_0(p, w_0) = \int_{\Omega} f_r(p, w_i, w_0) L_i(p, w_i) n \cdot w_i d_w [3]$$

The microfacet surface model

Any surface on a microscopic scale can be described by tiny perfectly reflective mirrors called microfacets. The alignment of these small mirrors can vary greatly depending on the roughness of a surface.

The more chaotically aligned each microfacet along a surface is, the rougher it is. When it comes to specular lighting/reflection, the effect of these tiny-like mirror alignments is that incoming light rays are more likely to scatter in completely different directions on rougher surfaces, resulting in a more widespread specular reflection. On the other hand, light rays are more likely to reflect in the same direction on a smooth surface, resulting in smaller and sharper reflections.

The idea that rough surfaces can be treated as a collection of microscopic microfacets is at the heart of several geometric-optics-based techniques to modeling surface reflection and transmission. Microfaceted surfaces are frequently modeled as heightfields, in which the distribution of facet orientations is statistically defined.

Microfacet-based BRDF models the scattering of light from a large number of microfacets statistically. If we suppose that the illuminated differential area is quite significant compared to the size of individual microfacets, then a large number of microfacets are lit, and their aggregate behavior determines the observed scattering.

The two fundamental components of microfacet models depict the distribution of facets and a BRDF that defines how light scatters from individual microfacets. The goal is to create a closed-form expression giving the BRDF that characterizes scattering from a surface. Although specular transmission is beneficial for simulating many translucent materials, and the Oren–Nayar model treats microfacets as Lambertian reflectors, perfect mirror reflection is most usually utilized for the microfacet BRDF.

Material parameters

With a working understanding of PBR's underlying mathematical model, the next step is to describe how artists can create in the Yūgen Engine the physical qualities of a surface directly fed into the PBR equations. Textures can define or represent all of the surface parameters required for a PBR workflow (Figure 10)

There are only four different properties that the Yugen Engine makes use of regarding the "physically based" component of its materials system. These are the following:

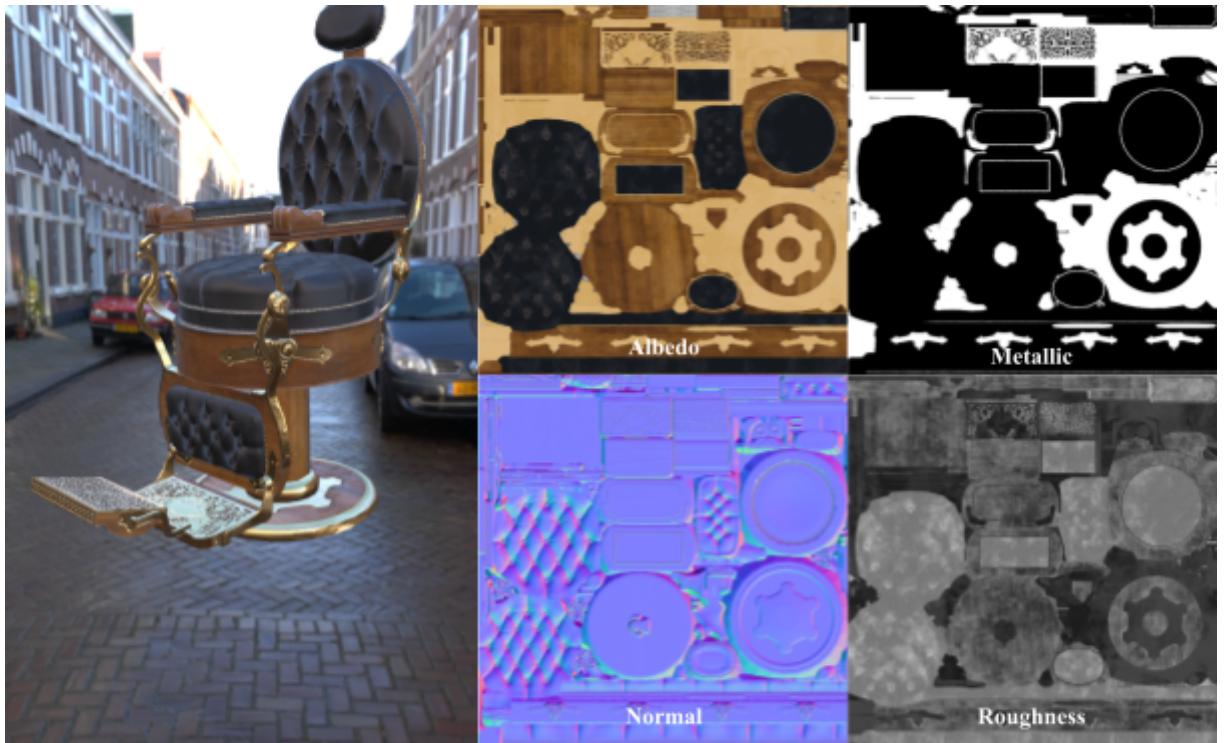


Figure 10: Scene rendered in Yūgen Engine with the 3D mesh and materials

- Base Color
- Normal
- Metallic
- Roughness

Base Color

The albedo texture defines the color of the surface for each texel, or the base reflectivity if the texel is metallic. This is comparable to a diffuse texture, but it extracts all illumination information from the texture. An albedo texture should only contain the color (or refracted absorption coefficients) of the surface; diffuse textures often have minor shadows or darkened fissures inside the image, whereas an albedo texture should only contain the color (or refracted absorption coefficients) of the surface.

Normal

The normal map allows developers to set a unique normal per fragment to make it appear as if a surface is bumpier than its flat equivalent.

Metallic

The metallic map indicates whether a texel is metallic or not per texel. Metalness can be created in either grayscale or binary black and white, depending on how the PBR engine is configured.

Roughness

The roughness map provides the roughness of a surface per texel. The statistical microfacet orientations of the surface are influenced by the sampled roughness value of the roughness. Reflections on a rougher surface are larger and blurrier, but reflections on a smooth surface are focused and clear. In Figure 11 it can be seen how reflections are displayed on metal (upper picture in Figure 11) and non-metal materials (lower picture in Figure 11) as the roughness coefficient increases.

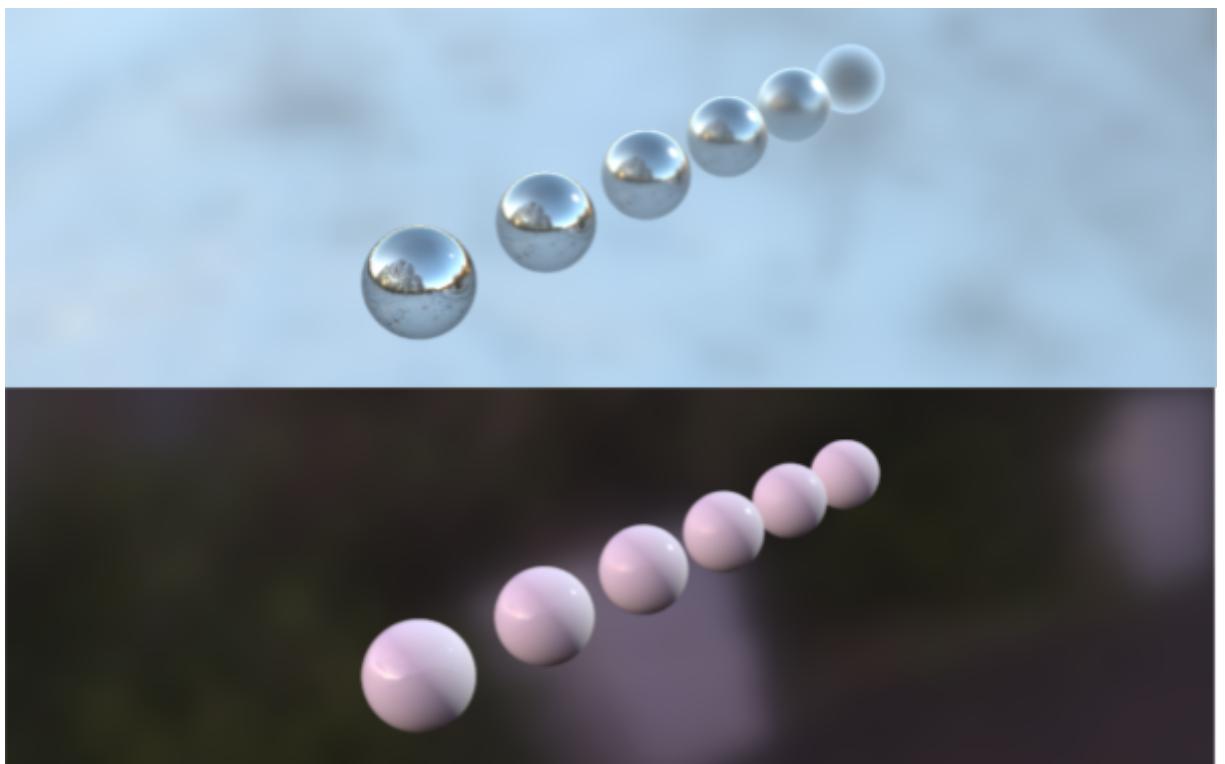


Figure 11: Scene rendered in Yūgen Engine

On a per-texel basis, artists can specify and alter these physically-based input values, and texture values can be based on the physical surface qualities of real-world materials. (Figure 12 shows different authored materials). One of the most significant benefits of a PBR render process is that these physical attributes of a surface remain constant regardless of environment or lighting arrangement, making it easier for artists to achieve physically realistic effects. Surfaces created using a PBR pipeline can be shared between different PBR render

engines, will look right regardless of the environment they are in and will appear more natural as a result.

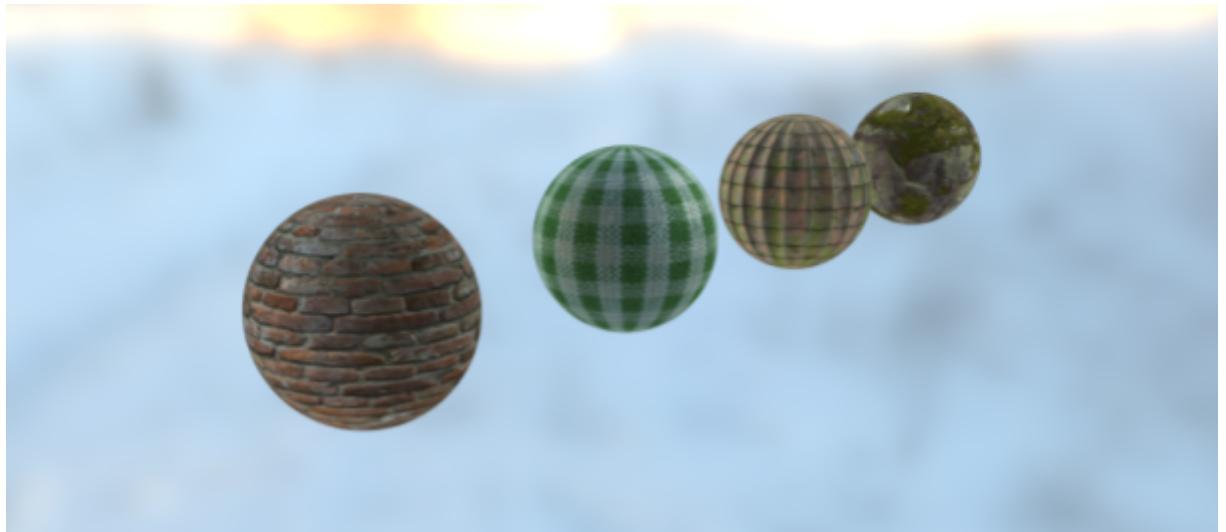


Figure 12: Scene rendered in Yūgen Engine

Comparisons between rendering in Yūgen Engine and Blender

The comparisons between Blender and Yūgen Engine can be seen in Figure 13 and Figure 14.

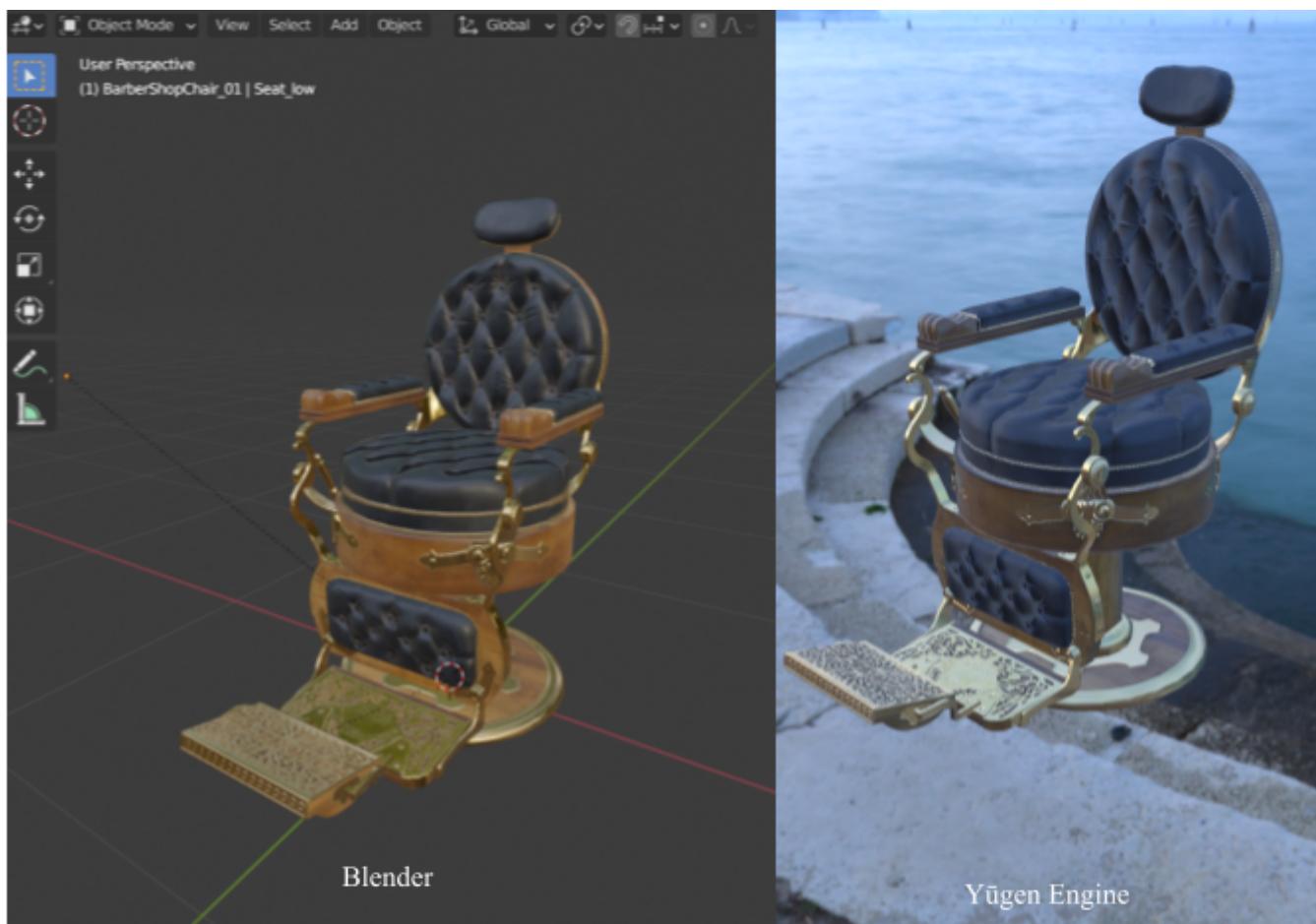


Figure 13

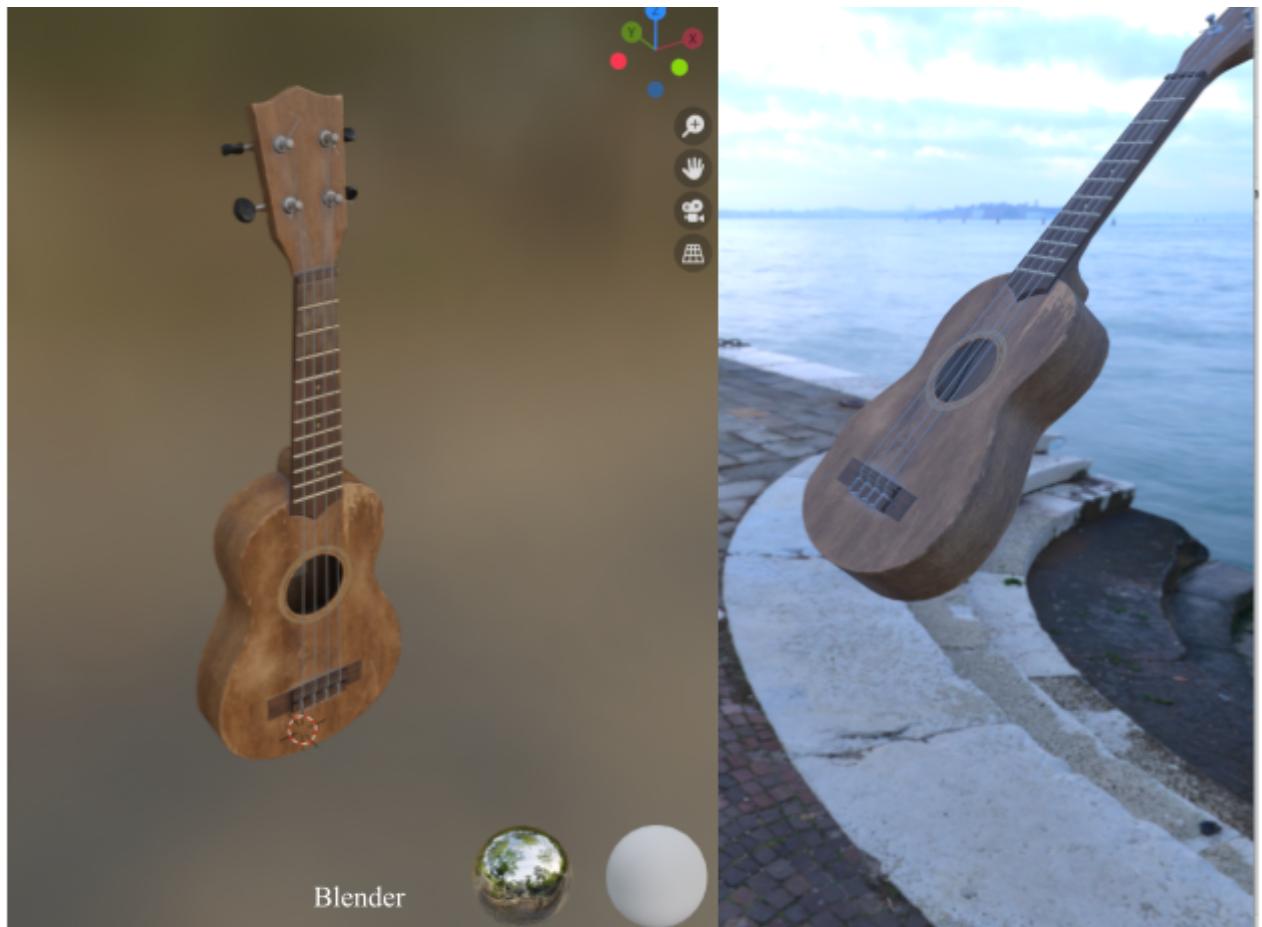


Figure 14

Natural Language Processing Pipeline

GPT-3 (Generative Pre-trained Transformer 3) is a deep learning-based autoregressive language model that produces human-like writing. The GPT-n series' third-generation language prediction model (and the successor to GPT-2⁵) was developed by OpenAI, a San Francisco-based artificial intelligence research group. GPT-3 has a total capacity of 175 billion machine learning parameters in its complete edition. GPT-3 is part of a trend in natural language processing (NLP) systems using pre-trained language representations, which was launched in May 2020 and was in beta testing as of July 2020. This model featured 100 times more parameters than GPT-2[4] and ten times more than Microsoft's sophisticated Turing NLG language model (Figure 15).

The architecture of GPT-3 is the same as GPT-2. Few major differences from GPT-2 are (Figure 16):

- GPT-3 has 96 layers, with each layer having 96 attention heads.
- The size of word embeddings was increased to 12888 for GPT-3 from 1600 for GPT-2.
- The context window size was increased from 1024 for GPT-2 to 2048 tokens for GPT-3.
- Adam optimiser was used with $\beta_1=0.9, \beta_2=0.95$ and $\epsilon=10^{-8}$.
- Alternating dense and locally banded sparse attention patterns were used

GPT-3 has a wide range of features, including the one that stands as the foundation for the NLP component of Yūgen Engine, which is the capacity to produce conditional synthetic text samples of unrivaled quality by priming the model with input and having it generate a long continuation. GPT-3 also outperforms other language models trained on specific domains (such as Wikipedia, news, or literature) without the requirement for domain-specific training datasets.

⁵ <https://github.com/openai/gpt-2>

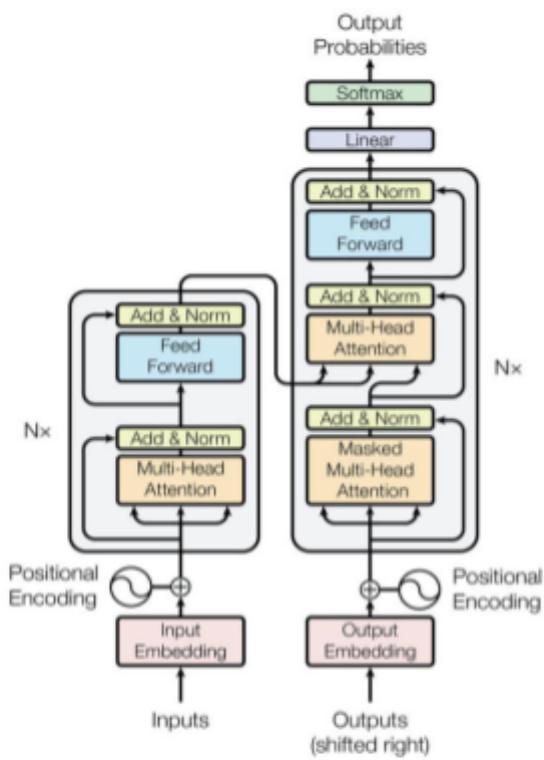


Figure 15

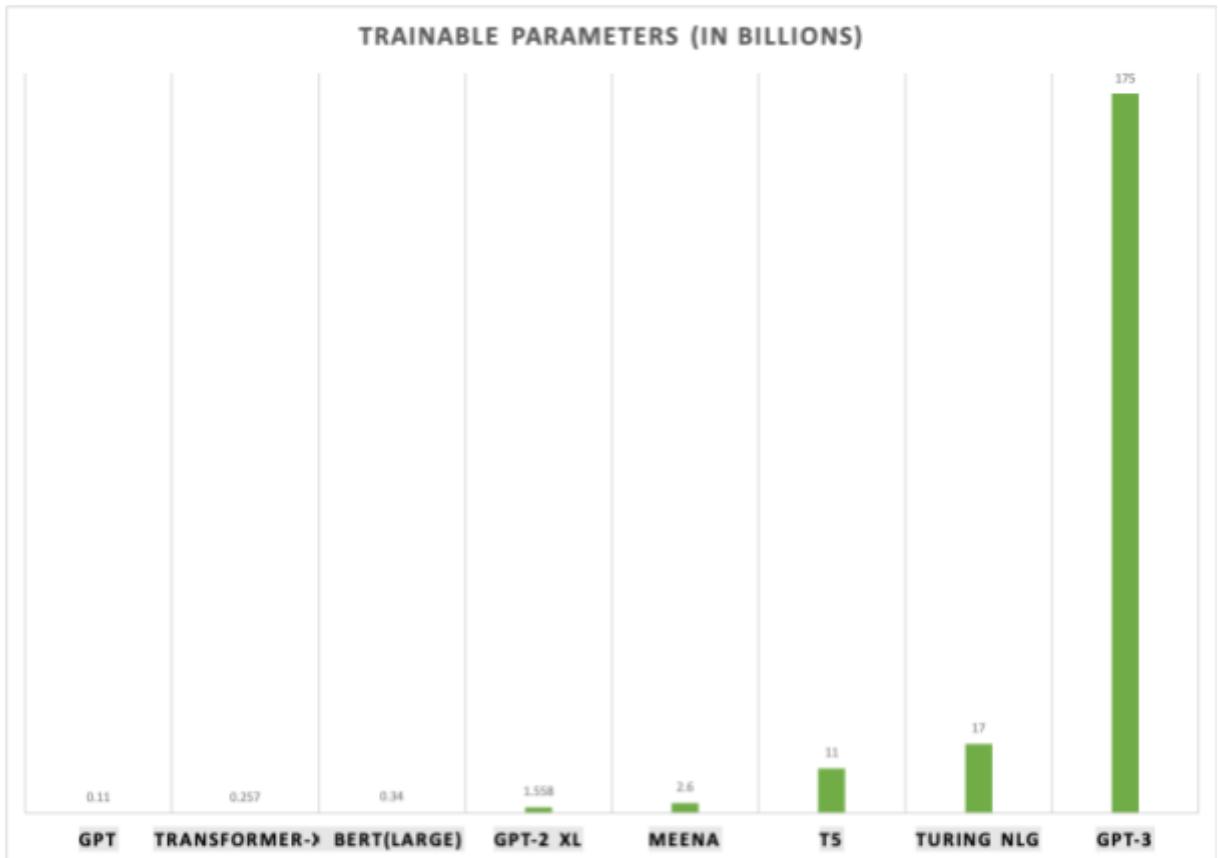


Figure 16

APIs

OpenGL

OpenGL⁶ is a software library that allows developers to access graphics hardware functions through an application programming interface. The OpenGL latest version (which the Yūgen Engine supports) has approximately 500 instructions for specifying the objects, pictures, and operations required to create interactive three-dimensional computer graphics programs.

OpenGL is a streamlined, hardware-independent interface that may be implemented in software (if no graphics hardware is present in the system) or entirely in hardware (if no graphics hardware is currently in the system), regardless of the computer's operating or windowing system.

The following list briefly describes the major operations that the Yūgen Engine would perform to render an image using OpenGL:

- Specify the data for constructing shapes from OpenGL's geometric primitives.
- Execute various shaders to perform calculations on the input primitives to determine their position, color, and other rendering attributes.
- Convert the mathematical description of the input primitives into their fragments associated with locations on the screen. This process is called rasterization.
- Finally, execute a fragment shader for each of the fragments generated by rasterization, which will determine the fragment's final color and position.
- Possibly perform additional per-fragment operations such as determining if the object that the fragment was generated from is visible or blending the fragment's color with the current color in that screen location.

The OpenAI API

The OpenAI API⁷ has a general-purpose "text in, text out" interface that can be used for almost any language task. Unlike most other language APIs, which are built for a specific

⁶ <https://www.opengl.org/>

⁷ <https://openai.com/blog/openai-api/>

task, such as sentiment categorization or named entity identification, this one is multi-purpose.

It is utilized in the Yūgen Engine API simply by presenting it with a text prompt (text-based input or "instructions"), and it will return a text completion, attempting to match the context or pattern provided. It can be "programmed" by writing a description or a few samples of what it should accomplish. The task's complexity determines the degree to which it succeeds. A good rule of thumb is thinking about writing out a word problem for a middle schooler to solve.

The API is built around three concepts: prompt, completion, and tokens. The "prompt" is the text entered into the API, and the "completion" is the text generated by the API based on the prompt. It is worth mentioning that the API is stochastic by default, which means that even if the prompt remains the same, one can get a slightly different completion each time one calls it. For example, if one asks the API, "As Shakespeare said, to be, or not," it will almost certainly return the completion "to be."

With various speed and throughput enhancements, the API executes models with weights from the GPT-3 family. The following models are now available in Yūgen Engine: davinci, curie, babbage, and ada. The versions offer a range of capabilities, with davinci being the most capable and ada being the fastest. When experimenting with Yūgen Engine's AI capabilities, it is recommended to use davinci to provide the most outstanding results. When one is ready to put one's generated game narratives into production, one shall experiment with the alternative models to see if one can get the same goals with less latency.

Case studies

Generating stories

Scenario 1:

The developer creates a 3D environment and wants to generate a game storyline. One chooses the category of interest, fantasy, and the role of The Noble and waits for the story to be generated. After, the game creator has complete liberty to generate an infinite interactive fictional scenario with the aid of GPT-3 (Figure 17).

Transcript of the AI generated story:

GPT-3: You are a noble living in the kingdom of Larion. You have a pouch of gold and a small dagger. You are awakened by one of your servants who tells you that your keep is under attack. You look out the window and see a large group of humanoid creatures riding strange beasts. They look strange, almost as if they aren't completely human. They also appear to be led by some sort of shaman.

PLAYER: Scared, I go to the royal witch for advice.

GPT-3: She tells me that they are monsters called orcs and that they are attacking the castle because the king has not paid them tribute. She tells me that I must go to the king and persuade him to pay them. I go to the king and tell him of the situation. He tells me that he will pay them the tribute, but that I must tell.

PLAYER: A fairy comes with a premonition and magical sword.

GPT-3: I tell the king of the vision and he tells me that I must lead the army into battle. I lead the army into battle and they easily defeat the orcs. The king tells me that I must go to the land of the orcs and kill their leader. I do so and I return to the king with the head of the orc.

PLAYER: The king is proud

...

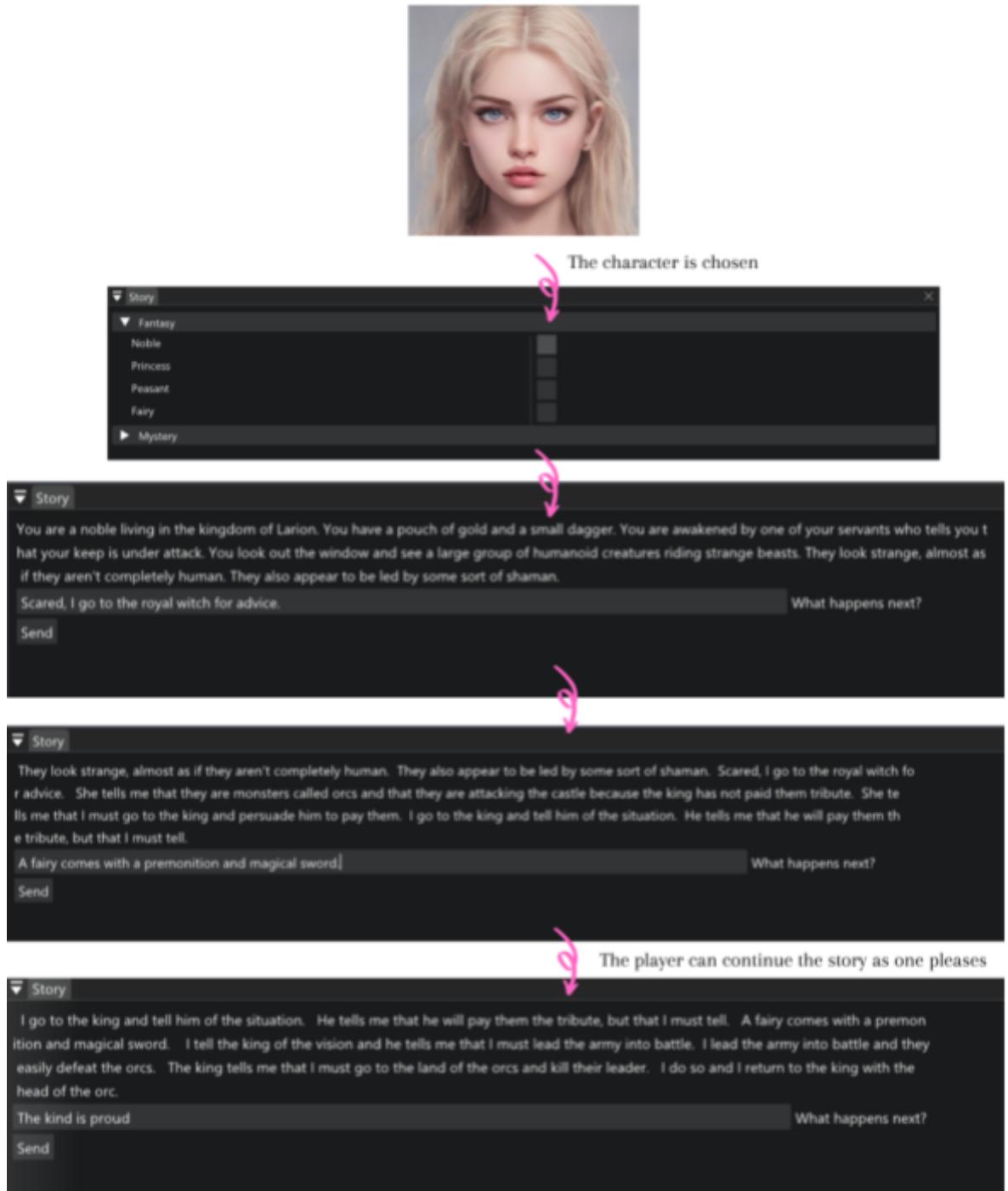


Figure 17

Scenario 2:

The developer chooses the category of interest, fantasy, and the role of The Fairy and waits for the story to be generated. Following that, using GPT-3, the game developer is free to create an endless interactive fictitious scenario (Figure 18).

Transcript:

GPT-3: You are a fairy living in the kingdom of Larion. You live in a fairy realm hidden under a grassy hill near the castle. Your skin is a light tinge of blue, your wings sparkle in the sunlight and you are very small and good at hiding. You are fascinated by humans and love to watch them from their flowers where you blend in.

PLAYER: While you are hiding, suddenly you notice a beautiful human girl. She is clearly searching for something. But what could it be?

GPT-3: You are determined to find out. Maybe it is something that could help you solve the mystery of the missing Prince, the heir to the throne of Larion.

PLAYER: She probably needs a magical potion made by a fairy but I am too scared to show my face.

GPT-3: You must help her find the ingredients for the potion.



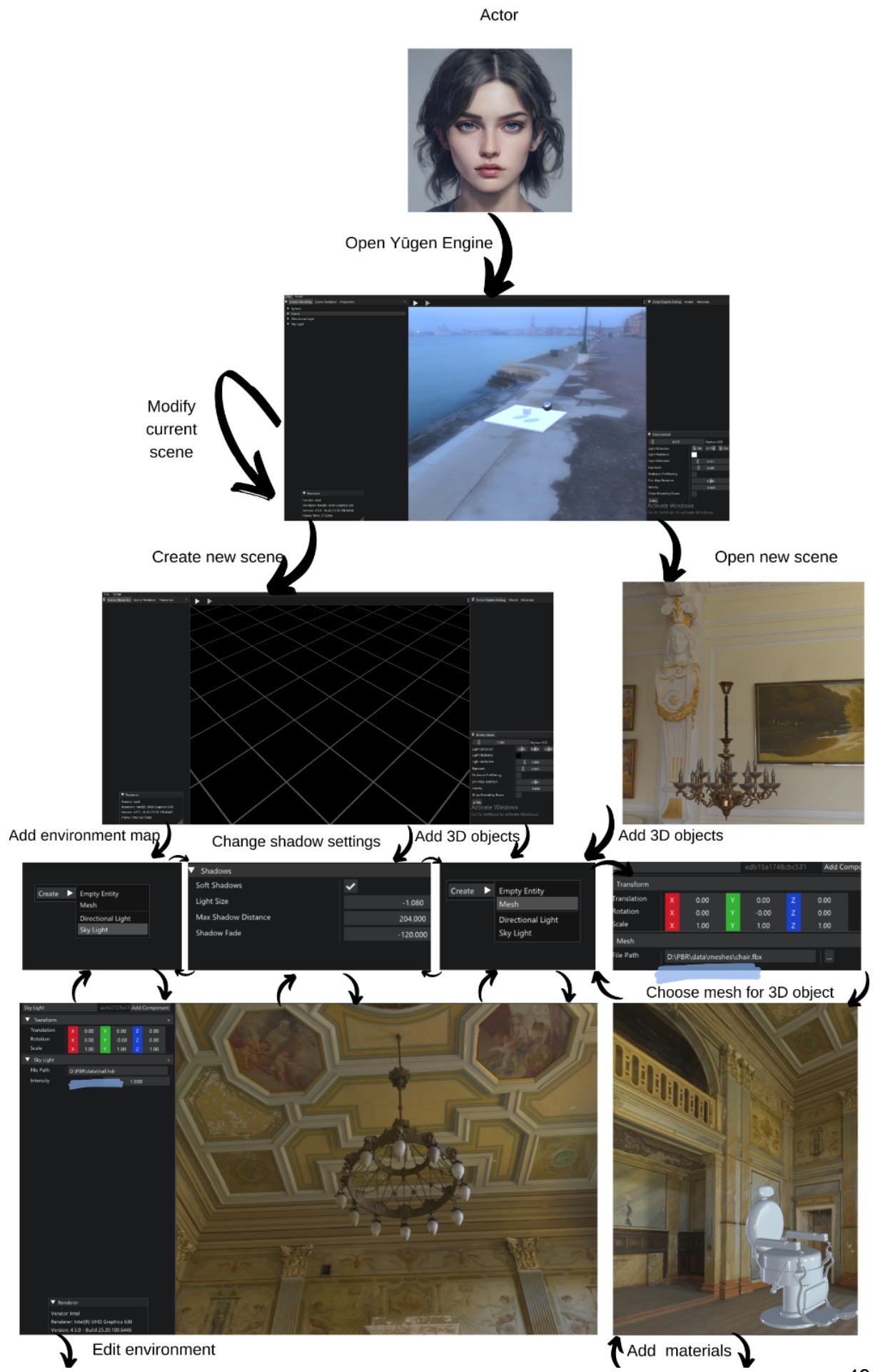
Figure 18

Creating photorealistic environments and characters

The following diagram (Figure 19) shows some of the mechanisms at the game developer's disposal to construct a hyper-realistic world.

When Yūgen starts, a scene is loaded into the engine. At this point, one can modify the current scene, load a new one previously created or start another one from scratch.

The scenes can be altered by importing 3D meshes and textures, changing the positions of the 3D models (rotation, translation, scaling), creating lights, loading environment maps, or modifying the environment parameters (such as light direction, light radiance, shadows, and much more).



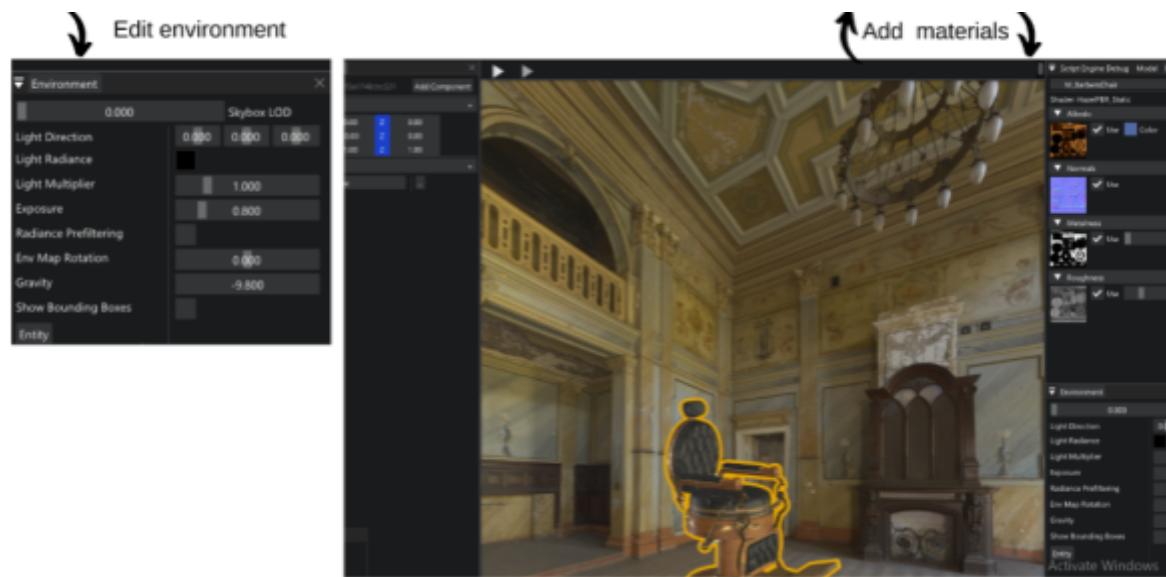


Figure 19

Conclusions

The Yūgen Engine's goal is the complete removal of pre-established responses and attitudes of NPCs and players in video games, letting the users explore endlessly, continuously finding an infinite amount of entirely new content, adventures, and original perspectives of different characters. The game engine proposed in this thesis has as the target users the game developers that are interested in creating a naturally fluid universe with the narratives being controlled by natural language processing technology.

One of the most significant features is that the players can express themselves naturally and have the action flow according to their attitudes or original actions without being confined by the game developer's vision. At this moment, the Yūgen Engine can be used to create open-world interactive fiction 3D rendering productions empowered by artificial intelligence.

Another relevant factor of the Yūgen Engine's usefulness constitutes its state-of-the-art rendering pipeline that uses the Unreal Engine 4 PBR notes, Frostbite's SIGGRAPH 2014 paper, and Disney's physically based shading and material model. Therefore, using Yūgen Engine graphics could create both immersive, photorealistic imagery and creative stylized worlds.

This paper also expresses a richness in the technologies used since game engine programming and NLP application development dominate most innovative AI markets. The motivation for this can be found in the dedicated chapter.

Future Improvements

Another ambition would be offering NPCs unique, individual personalities that would evolve and would powerfully impact characters' answers concerning their value system and conscience.

This technology is still new, and it is a source of excitement to see where it leads.

Bibliography

1. Matt Pharr, Greg Humphreys, "Physically Based Rendering: From Theory to Implementation, 2nd ed.", 2010

URL:<https://www.amazon.com/Physically-Based-Rendering-Second-Implementation/dp/0123750792>

2. Brian Karis, Epic Games, "Unreal Engine 4 PBR notes", 2013

URL:https://blog.selfshadow.com/publications/s2013-shading-course/karis/s2013_pbs_epic_notes_v2.pdf

3. Joey de Vries, Learn OpenGL: Learn modern OpenGL graphics programming in a step-by-step fashion, 2020

URL: <https://learnopengl.com/PBR/Theory>

4. Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, Language Models are Unsupervised Multitask Learners, 2019

URL:

https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf