ДОДАТОК А

ПРИКЛАД ПРОГРАМНОГО КОДУ

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Screenshooter: MonoBehaviour {
    private bool ssEnabled = false;
    private bool loopEnabled = false;
    private string savePath = "";

    void Start() {
        if (GetEnvironmentVariable("IMGEN_TAKE_SCREENSHOTS", "") == "1") {
            ssEnabled = true;
        }
        if (GetEnvironmentVariable("IMGEN_LOOP", "") == "1") {
            loopEnabled = true;
        }
        savePath = GetEnvironmentVariable(
            "IMGEN_SCREENSHOT_PATH",
            System.Environment.CurrentDirectory
        );

        Debug.Log("Take screenshots = " + ssEnabled);
        Debug.Log("Loop mode = " + loopEnabled);
        Debug.Log("Screenshot directory = " + savePath);

        if (loopEnabled) {
            StartCoroutine(Loop());
        }
    }

    private string GetEnvironmentVariable(string key, string defaultValue) {
        var envs = Environment.GetEnvironmentVariables();
        if (envs.Contains(key)) {
            return envs[key].ToString();
        }
        return defaultValue;
    }

    private IEnumerator Loop() {
        for (var i = 0; i < 10; i++) {
            Debug.Log("Iteration " + i);
            if (ssEnabled) {
                CaptureNamedScreenshot();
            }
            Debug.Log("Done.");
            yield
            return new WaitForSeconds(1);
        }
        #if UNITY_EDITOR
        UnityEditor.EditorApplication.isPlaying = false;
        #else
        Application.Quit();
        #endif
    }

    private void CaptureNamedScreenshot() {
        var name = GetFilename();
        Debug.Log("Writing screenshot: " + name);
        ScreenCapture.CaptureScreenshot(name);
    }

    private string GetFilename() {
```

```
        return string.Format(
            "{0}/img_{1}.png",
            savePath,
            System.DateTime.Now.ToString("yyyy-MM-dd_HH-mm-ss-fff")
        );
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ObjectMover: MonoBehaviour {
    public float rotationX = 14 f;
    public float rotationZ = -10 f;
    private float smooth = 5.0 f;
    private Quaternion target;

    void Start() {
        StartCoroutine(Loop());
    }

    private IEnumerator Loop() {
        while (true) {
            DetermineNewRotation();
            yield
            return new WaitForSeconds(1);
        }
    }

    private void DetermineNewRotation() {
        target = Quaternion.Euler(rotationX, Random.Range(-360.0 f, 360.0 f),
rotationZ);
    }

    void Update() {
        transform.rotation = Quaternion.Slerp(transform.rotation, target,
Time.deltaTime * smooth);
    }
}

Shader "Hidden/OpticalFlow"
{
    Properties
    {
        _Sensitivity("Sensitivity", Float) = 1
    }
    SubShader
    {
        // No culling or depth
        Cull Off ZWrite Off ZTest Always

        Pass
        {
            CGPROGRAM
            #pragma vertex vert
            #pragma fragment frag

            #include "UnityCG.cginc"

            struct appdata
```

```
            {
                float4 vertex : POSITION;
                float2 uv : TEXCOORD0;
            };

            struct v2f
            {
                float2 uv : TEXCOORD0;
                float4 vertex : SV_POSITION;
            };

            float4 _CameraMotionVectorsTexture_ST;

            v2f vert (appdata v)
            {
                v2f o;
                o.vertex = mul(UNITY_MATRIX_MVP, v.vertex);
                o.uv = TRANSFORM_TEX(v.uv, _CameraMotionVectorsTexture);
                return o;
            }
            sampler2D _CameraMotionVectorsTexture;

            float3 Hue(float H)
            {
                float R = abs(H * 6 - 3) - 1;
                float G = 2 - abs(H * 6 - 2);
                float B = 2 - abs(H * 6 - 4);
                return saturate(float3(R,G,B));
            }

            float3 HSVtoRGB(float3 HSV)
            {
                return float3(((Hue(HSV.x) - 1) * HSV.y + 1) * HSV.z);
            }

            float _Sensitivity;

            float3 MotionVectorsToOpticalFlow(float2 motion)
            {
                float angle = atan2(-motion.y, -motion.x);
                float hue = angle / (UNITY_PI * 2.0) + 0.5;// convert motion
angle to Hue
                float value = length(motion) * _Sensitivity;// convert
motion strength to Value
                return HSVtoRGB(float3(hue, 1, value));// HSV -> RGB
            }

            fixed4 frag (v2f i) : SV_Target
            {
                float2 motion = tex2D(_CameraMotionVectorsTexture, i.uv).rg;
                float3 rgb = MotionVectorsToOpticalFlow(motion);
                return float4(rgb, 1);
            }
            ENDCG
        }
    }
}
```

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Screenshooter: MonoBehaviour {
    private bool ssEnabled = false;
    private bool loopEnabled = false;
    private string savePath = "";

    void Start() {
        if (GetEnvironmentVariable("IMGEN_TAKE_SCREENSHOTS", "") == "1") {
            ssEnabled = true;
        }
        if (GetEnvironmentVariable("IMGEN_LOOP", "") == "1") {
            loopEnabled = true;
        }
        savePath = GetEnvironmentVariable(
            "IMGEN_SCREENSHOT_PATH",
            System.Environment.CurrentDirectory
        );

        Debug.Log("Take screenshots = " + ssEnabled);
        Debug.Log("Loop mode = " + loopEnabled);
        Debug.Log("Screenshot directory = " + savePath);

        if (loopEnabled) {
            StartCoroutine(Loop());
        }
    }

    private string GetEnvironmentVariable(string key, string defaultValue)
{
        var envs = Environment.GetEnvironmentVariables();
        if (envs.Contains(key)) {
            return envs[key].ToString();
        }
        return defaultValue;
    }

    private IEnumerator Loop() {
        for (var i = 0; i < 10; i++) {
            Debug.Log("Iteration " + i);
            if (ssEnabled) {
                CaptureNamedScreenshot();
            }
            Debug.Log("Done.");
            yield
            return new WaitForSeconds(1);
        }
        #if UNITY_EDITOR
        UnityEditor.EditorApplication.isPlaying = false;
        #else
        Application.Quit();
        #endif
    }
```

```
    private void CaptureNamedScreenshot() {
        var name = GetFilename();
        Debug.Log("Writing screenshot: " + name);
        ScreenCapture.CaptureScreenshot(name);
    }

    private string GetFilename() {
        return string.Format(
            "{0}/img_{1}.png",
            savePath,
            System.DateTime.Now.ToString("yyyy-MM-dd_HH-mm-ss-fff")
        );
    }
}

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ObjectMover: MonoBehaviour {
    public float rotationX = 14 f;
    public float rotationZ = -10 f;
    private float smooth = 5.0 f;
    private Quaternion target;

    void Start() {
        StartCoroutine(Loop());
    }

    private IEnumerator Loop() {
        while (true) {
            DetermineNewRotation();
            yield
            return new WaitForSeconds(1);
        }
    }

    private void DetermineNewRotation() {
        target = Quaternion.Euler(rotationX, Random.Range(-360.0 f, 360.0
f), rotationZ);
    }

    void Update() {
        transform.rotation = Quaternion.Slerp(transform.rotation, target,
Time.deltaTime * smooth);
    }
}
```

ДОДАТОК Б

СЛАЙДИ ПРЕЗЕНТАЦІЇ

Міністерство освіти і науки України

Харківський національний університет радіоелектроніки

Атестаційна робота магістра

Дослідження та оптимізація алгоритмів створення, рендерінгу та обміну синтетичними даними з системами машинного навчання

Науковий керівник: Вечур О. В.
к.т.н., доц. каф. ПІ

Виконала: Тихонова О.О
студентка групи ІПЗмзд-18-1

2020

1

Мета дослідження

- Дослідити та систематизувати можливі аспекти впливу на результати навчання за синтетичними даними, які були створені за допомогою ігрових движків.
- Розглянути можливості та переваги ігрових движків у контексті створення синтетичних даних.
- Дослідити можливий інтерфейс взаємодії між системою генерації та системами машинного навчання, а також методи оптимізації цього процесу.

2

Практична релевантність та новизна

Практичним значенням цього дослідження є можливість емулювати велику кількість даних для вирішення задачі навчання систем, які прогнозують певні події.

У цьому контексті, новизною даного дослідження є використання ігрових движків для створення великої кількості даних, які неможливо відтворити вручну. Це дозволить використовувати штучний інтелект та машине навчання у більшій кількості доменів життєдіяльності.

3

Синтетичні дані

Синтетичні дані – це штучно створена інформація за допомогою певних алгоритмів та обмежень, а не з реальних подій.

Переваги:

- низька ціна;
- контрольованість;
- лімітовані тільки ступенем випадковості.

Недоліки:

- недостатній реалізм.

4

Фактори впливу на результати генерації

- кількість елементів у сцені;
- кількість полігонів у моделях;
- деталізованість текстур;
- формати використаних текстур та моделей (PNG, TGA, JPEG, OBJ, FBX);
- використання відео чи зображень;
- використання PBR текстур;
- використовування hand–paint текстур;
- налаштування параметрів світла – baked vs real time;
- кількість джерел світла;
- та інші…

5

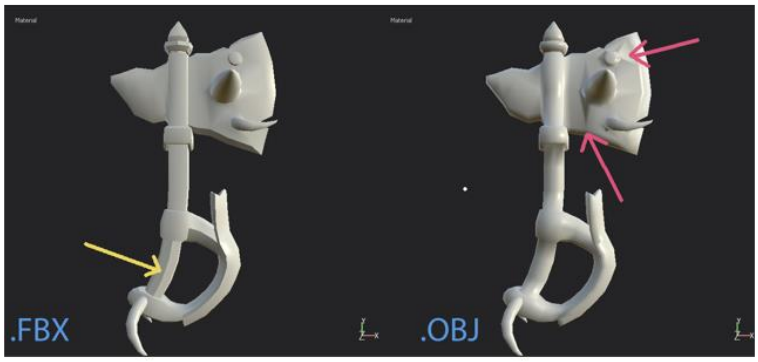
Інструменти

Ігрові движки - ідеальні інструменти для створення фізично-реалістичних симуляцій реального світу. Вони дозволяють легко контролювати те як виглядає сцена, динамічно змінювати налаштування. Тому вони також можуть бути використані для створення штучних даних.

6

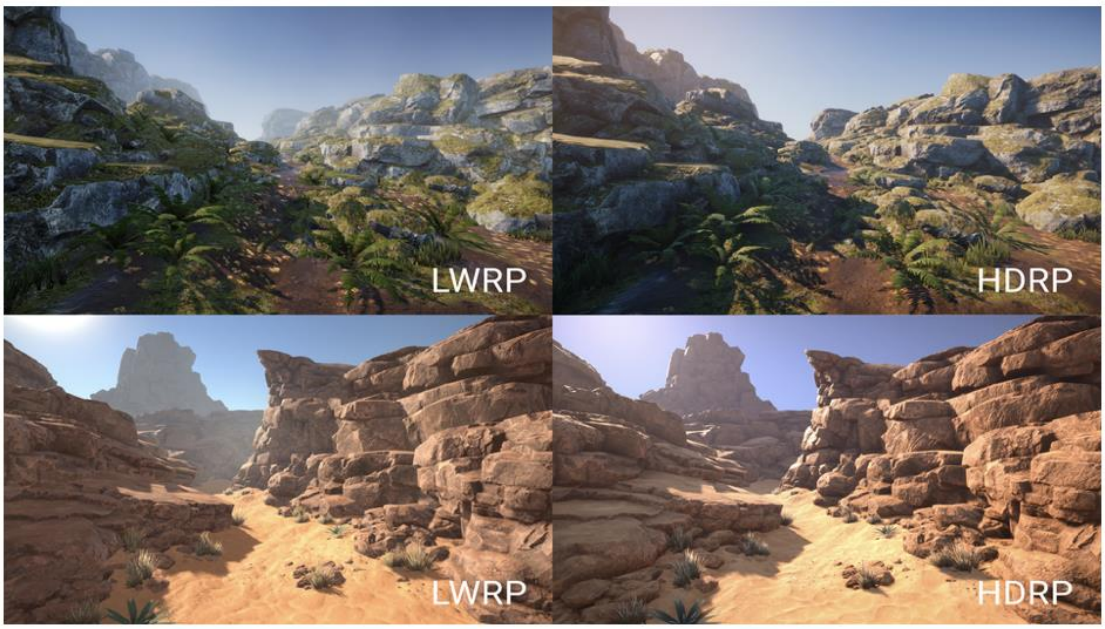
Об’єкти обробки

- моделі (FBX, OBJ);
- текстури (PNG, JPEG, TGA);
- мета дані та конфігурації;
- відео - MP4, FLV, AVI.



7

HDRP та URP



8

Текстури та матеріали

- PBR (Physical based rendering) та hand paint;
- Height, Normal, Roughness, Albedo - мають бути присутні незалежно від типу задачі;
- Ambient Occlusion - можна не використовувати при низькій кількості ерозійних об’єктів.

9

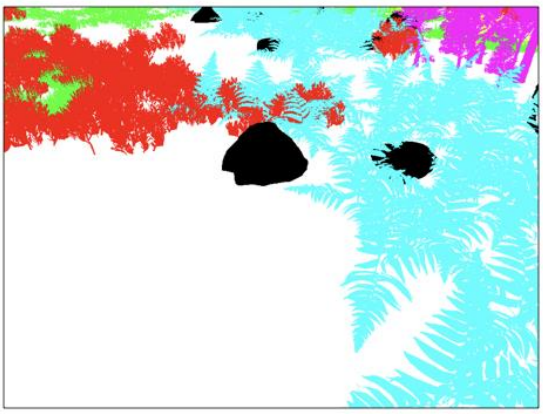
Ефекти пост-обробки, освітлення сцени

10

Системи частинок - VFX Graph та класична

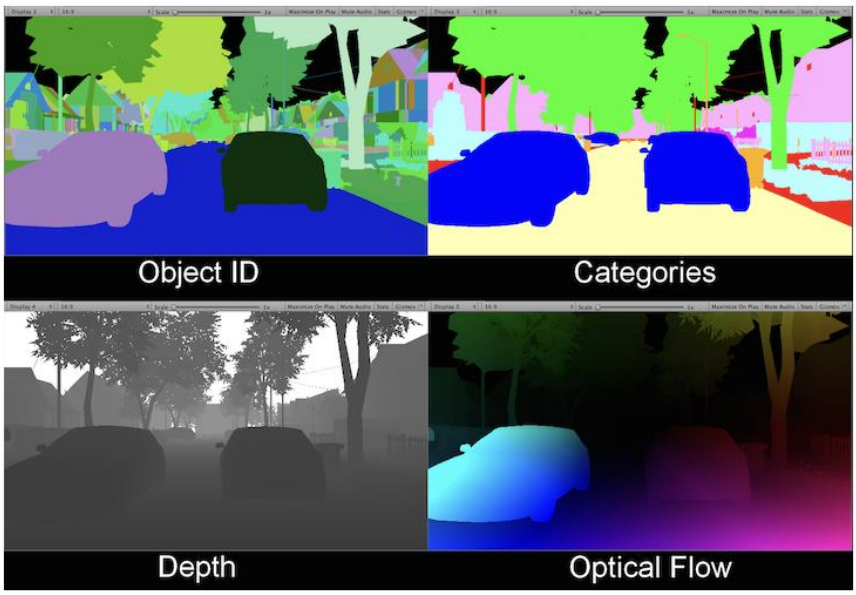
11

Сегментація зображень



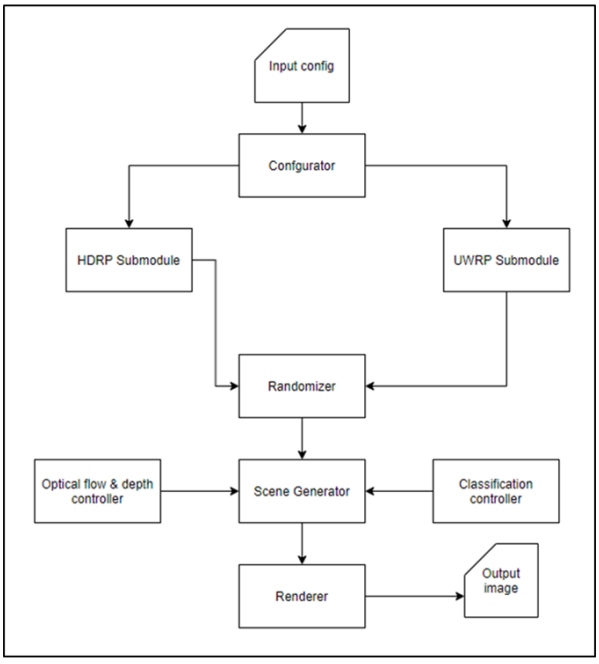
12

Категорізація, оптичний потік та глибина



13

Проектування модулів програмної системи



14

Приклад коду - випадкове обертання об’єктів

```
public class ObjectMover: MonoBehaviour {
    public float rotationX = 14 f;
    public float rotationZ = -10 f;
    private float smooth = 5.0 f;
    private Quaternion target;

    void Start() {
        StartCoroutine(Loop());
    }

    private IEnumerator Loop() {
        while (true) {
            DetermineNewRotation();
            Yield return new WaitForSeconds(1);
        }
    }

    private void DetermineNewRotation() {
        target = Quaternion.Euler(rotationX, Random.Range(-360.0 f, 360.0 f), rotationZ);
    }

    void Update() {
        transform.rotation = Quaternion.SIerp(transform.rotation, target, Time.deltaTime * smooth);
    }
}
```

15

Практичне застосування

16

Висновки

В ході дослідження:

- знайдено нові підходи до генераціх синтетичних даних з використанням ігрових движків;
- виділено основні властивості сцен, котрі можуть впливати на результати навчання;
- запропоновано методи оптимізації ренедрінгу та генерації синтетичних зображень та відео;
- виконано апробацію роботи у вигляді тезисів до конференції «Наукова думка сучасності і майбутнього».

17

ДОДАТОК В

АПРОБАЦІЯ РЕЗУЛЬТАТІВ РОБОТИ

СТВОРЕННЯ ТА РЕНДЕРІНГ СИНТЕТИЧНИХ ДАНИХ У ІГРОВИХ ДВИЖКАХ

naukam.triada.in.ua

Тихонова Олена Олександрівна

Студентка кафедри Програмної Інженерії

Харківський Національний Університет Радіоелектроніки

м. Харків

Вилегжанін Станіслав Сергійович

Студент кафедри Програмної Інженерії

Харківський Національний Університет Радіоелектроніки

м. Харків

Анотація: *Розгляд використання синтетичних даних, створених у ігрових движках для використанні при машинному навчанні.*

Ключові слова: *Синтетичний, штучний, зображення, текстура, карта, движок.*

З кожним днем сучасний світ змінюється та перетворюється. Технології «еволюціонують» та становляться все більш адаптовані під потреби конкретного користувача. Головним інструментом створення індивідуалізованого програмного і апаратного забезпечення стають системи машинного навчання та штучного інтелекту. Однак, щоб ефективно ними користуватися – потрібні дані, які б могли бути застосовані для вирішення поставленої задачі. Трапляється так, що виникають такі задачі, де дані для навчання систем, які зможуть їх вирішити, – або взагалі відсутні, або їх отримання може коштувати значних ресурсів. Прикладом однієї з таких задач може бути створення системи, яка буде аналізувати стан помешкання та визначати чи є загроза пожежі, чи ні. Для цього нам потрібні відео, власне з пожежами, але здобути їх у достатній кількості не є можливим, а відтворювати власноруч – дорого та непрактично. Механізмом, який допоможе розробнику подібних систем, стають інструменти генерації синтетичних даних. Синтетичні дані – це штучно створена інформація за допомогою певних алгоритмів та лімітів, а не з реальних подій. У рамках цієї роботи, буде розглянуто проблеми, інструменти та способи створення саме візуальних (зображення, відео) синтетичних даних. Якщо переглянути всі існуючі програмні продукти у домені графічних редакторів, систем рендерінгу, пакетів 3D моделювання та ігрових движків, найбільш яскравими виступають саме останні. Причин цьому велика кількість:

- можливість процедурно створювати сцени будь-якої складності;
- використання декількох камер та сесій одночасно;
- можливості зручно та динамічно налаштовувати параметри рендерінгу, світла та камери;
- широкий спектр оптимізаційних рішень для пошвидшення роботи;
- реалістична поведінка світла у сцені (PBR);
- ефекти пост-обробки для камери;
- створення ефектів з використанням GPU.

Тобто сучасні ігрові движки, які використовуються для створення фізично–реалістичних симуляцій у кіно та при створенні ігор – також ідеальні для створення псевдовипадкових сцен, бо мають можливості гнучко налаштовувати різноманітні параметри. Це дозволяє кожен раз генерувати нову ситуацію, що дає можливість наближатися до реалістичних результатів машинного навчання та покращувати точність прогнозів. Беручи до уваги актуальність цієї проблематики та релевантність ігрових движків та систем рендерінгу у її вирішенні, метою даних тез є пошук кореляцій різноманітних параметрів налаштування сцен та їх рендерінгу у ігровому движку при генерації синтетичних даних, а також рішень на етапі моделювання та текстурування з точністю прогнозів системи машинного навчання. Другою задачею є порівняння різних движків у цьому аспекті, а також оптимізація процесу генерації даних під певну ситуацію. Для безпосередньої перевірки результатів, є релевантним використанням популярних та перевірених часом систем та алгоритмічних баз, наприклад таких, як TensorFlow, які використовуються у великій кількості в сучасних проектах. Якщо результати покажуть достатню ефективність системи, то це доведе можливість створення більш точних моделей у випадку більш спеціалізованих програмних пакетів для машинного навчання.

Слід зазначити, що елементами наукової новизни у цьому питанні є систематизовані та зважені аспекти, які впливають на результати машинного навчання при створенні синтетичних даних, а також створення технік оптимізацій цього процесу. Практичним значенням цього нарямку є можливість впровадити системи штучного інтелекту та машинного навчання у такі домени життя, в яких наразі це неможливо. А також, застосування у наукових дослідженнях в певних галузях, покращення тестування вже існуючих програмних забезпечень та продуктів, яке дозволить розширити можливості розробників у створенні власних прототипів у обраній галузі з реального життя.

Основна проблема синтетичних наборів даних – це наблизити їх до подібності з реальним випадком використання, особливо

відео. Але справжні проблеми виникають, коли прогнози зосереджуються на дрібних деталях, таких як розпізнавання обличчя. Крім того, рендерінг довгих фотореалістичних відео або зображень може бути надзвичайно важкою операцією. Але це можна подолати, наприклад, згенерувавши карти сегментації. Моделі, які можуть бути навчені синтетичними даними, обмежені насиченістю та «багатством» даних, які ми можемо отримати. Що стосується зображень, які створені за допомогою програмного забезпечення для 3D рендерінгу, то є можливість повторення значної частини реального зображення завдяки науковому розумінню світла та фізики.

Розглянемо також деякі інші способи генерації візуальних синтетичних даних, окрім використання ігрових движків, щоб мати більш повну картину переваг саме цього методу. Вирізання та вставка одного зображення в інше виглядає як перспективна ідея в цьому плані. У цьому аспекті є можливість вирізати набір предметів та підставляти їх до різноманітних оточень. Проблема цього механізму полягає в тому, що це надзвичайно важко вирізати та вставити предмет достатньо природнім та реалістичним виглядом; тому це потребує додаткових модифікації зображення методами аугментації та вдосконалення. Це н надто важливо розміщувати об’єкти природньо в глобальному масштабі зображення, але важливо досягти локального реалізму. Сучасні класифікатори об’єктів не так критично, щоб мати об’єкт, наприклад на столі чи на підлозі; однак важливо змішати об’єкт якомога реалістичніше на локальному тлі. Навіть досить наївна вставка об’єктів може допомогти покращити виявлення об’єктів у системах машинного навчання, зробивши по суті синтетичні дані. Наступним кроком у цьому напрямку було б створити вставлені об’єкти бути узгодженими з геометрією та іншими властивостями сцени. У цьому контексті є такі окремі випадки, як наприклад – локалізація тексту, тобто виявлення об’єктів спеціально для тексту, що з’являється на зображенні. Однак, усі ці методи виглядають не достатньо ефективними та швидкими, адже потребують дуже специфічних підходів до їх імплементації. Саме

тому будемо розглядати саме ігрові движки у якості основного інструмента.

Отже, першочерговою задачею є вибір ігрового движка для створення тестових даних. Буде розглянуто лише open-source чи умовно безкоштовні рішення, бо доступ та ліцензії до комерційних та внутрішніх движків різних компаній відсутні. Движки, які працюють лише з 2D графікою (cocos2dx, godot та інші) у дослідженні розглядаться не будуть, бо поставлена задача полягає у розгляненні саме залежності фотореалістичних ефектів до результатів навчання моделей.

Основними рішеннями на ринку рішень для створення ігор є Unreal Engine 4 (далі – UE4, Unreal) та Unity. Ці движки є основними конкурентами та знаходяться у перегонах, хто з них може називатися найбільш developer-friendly. Вони дуже схожі за набором можливостей, але реалізація та пріоритети у них досить різні. Unity є рішенням більш загального призначення з точки зору крос-платформності, що не є вирішальним фактором у роглянутому випадку. UE4 -сфокусований на пристрої високого класу (PC, консолі PS та XBOX). Також, Unreal зарекомендував себе у створенні VR програм та візуальних ефектів у кіноіндустрії. Однак, однією з метрик за якою будуть оцінюватись результати є час рендерінгу та обробки зображення, тому Unity може бути швидшим у створенні даних. У цьому випадку UE4 може стати не гнучким та нерелевантним [3].

Отже, виділяємо метрики за якими будемо оцінювати навчання моделей з синтетични даними:

- точність прогнозу;
- девальвація від навчання моделей за реальними даними;
- швидкість створення даних;
- швидкість рендерінгу;
- розмір створених файлів;
- розмір одного «instance» для роботи системи (розмір вихідного коду движків може займати велику кількість місця).

Наступним етапом є виділення параметрів, які можуть впливати на результати у рамках доступних функцій движка, таких як:

- формати використаних моделей (OBJ, FBX тощо);
- деталізованість текстур;
- UV розгортка;
- розширення текстур;
- формати використаних текстур (PNG, TGA, JPEG);
- використання відео чи зображень;
- адаптивність під різні типи задач;
- використання PBR текстур;
- налаштування параметрів світла – Baked vs Real time;
- кількість джерел світла;
- налаштування постпроцесингу камери;
- налаштування позиції камери (зум, кути огляду, ширина об'єктиву);
- використання ефектів часток (підрахування на CPU / GPU);
- використовування різних пайплайнів рендерінгу (у Unity, наприклад, HDRP – High Definition Render Pipeline, LWRP – Low Weight Render Pipeline, UWRP – Universal Render Pipeline);
- параметри налаштування матеріалів.

З точки зору системи навчання моделі для класифікації зображення, розглянемо такі можливості та параметри:

- швидкість навчання;
- стиснення зображень;
- точність прогнозів (у порівнянні з іншими системами);
- можливість використовувати згорткові нейронні мережі;
- можливість сегментації зображень;
- можливість візуалізувати результати.

Розглянемо одне з цих питань. Динамічна зміна кількості полігонів можлива лише за наявністю декількох однакових моделей з різними рівнями деталізації. У загальному сенсі кількість полігонів не є важливим фактором з точки зору

класифікації зображення, бо навіть для низькополігональних моделей основна деталізація доводиться саме на текстури. Тому цей показник можна вважати вторинним. Деталізованість текстур визначається наближенням до вигляду з реального світу. Це досягається за допомогою використання PBR (Physically Based Rendering) – набору технік візуалізації, в основі яких лежить теорія, наближена до реальної теорії поширення світла.Для того, щоб рендер движка міг інтерпретувати текстуру,як фізично коректну, потрібно створити не тільки карту кольору (альбедо), але ще й карту нормалей, карту жорсткості, карту висоти, карту металевості та карту оклюзії. Аналіз та рендерінг з використанням цих карт займає набагато більше часу та пам'яті, ніж з використанням hand–paint, де світло та тіні вмальовуються на пряму в карту кольору. Але, якщо є потреба в саме реалістичному зображенні, то іншого способу, окрім використання PBR, у даний момент немає. Тому, слід проаналізувати етапи та структуру фізично коректного рендерінгу та знайти можливі оптимізації які можуть знадобитися у ході дослідження.Текстура (карта) альбедо визначає для кожного пікселя колір поверхні або базову відбивну здатність, якщо цей піксель металевий. Карта нормалей дозволяє поставити для кожного фрагмента поверхні свою нормаль, щоб створити ілюзію, що ця поверхня більш випукла. Зазвичай усі інші карти є похідними від карти нормалей та можуть бути згенеровані за її допомоги, бо саме вона надає основну візуальну форму та деталізацію. Карта жорсткості вказує, наскільки жорстка поверхня знаходиться в основі пікселя текстури. Обране значення з текстури жорсткості визначає статистичну орієнтацію мікрограней поверхні. На жорсткішій поверхні виходять більш широкі і розмиті відображення, тоді як на гладкій поверхні вони сфокусовані та чіткі. Карта жорсткості інтерпретується у якості хаотичного розподілу мікрограней серед яких частина світла поглинається. Можна сказати, що ці мікрограні представлені у якості маленьких зеркал серед яких розсіюється вхідний потік світла. Результатом такого розміщення є те, що промені світла розсіюються в різних напрямках на жорсткіших поверхнях, що призводить до більш яскравого блиску. І навпаки на гладких поверхнях променій з більшою вірогідністю відіб'ються в одному

або паралельних напрямках, що дасть менш різкий відблиск. Карта металевості визначає, чи є піксель металевим, чи ні. Залежно від того, як налаштований PBR движок, можна задавати металічність, як відтінками сірого, так і тільки двома кольорами: чорним та білим. Зазвичай ця карта визначає, то наскільки сильно об'єкт може відзеркалювати світло. Чим темніший піксель тим більше світла він відбиває, надаючи поверхні блиску. Карта висоти (також відома як відображення паралакса) є концепцією, аналогічною відображенню нормалей, однак цей метод є більш складним – та, отже, більш дорогим. Такі карти зазвичай використовуються разом з картами нормалей, та часто вони застосовуються для надання додаткової чіткості поверхням [4].

Таким чином були розглянуті методи створення синтетичних даних за допомогою ігрових движків, а також можливі фактори впливу на результати машинного навчання у інших системах в контексті налаштувань такої системи генерації.

Література:

1.Medium [Електронний ресурс] / Medium – Режим доступу. https://medium.com – Загол. з екрану (дата звернення: 19.03.2020).

2.Unity Docs [Електронний ресурс] / Unity Docs – Режим доступу. https://docs.unity3d.com/Manual/ – Загол. з екрану (дата звернення: 16.03.2020).

3.Pharr, Physically Based Rendering, Third Edition: From Theory to Implementation [Текст] / Matt Pharr – Morgan Kaufmann, 2016 – 1266 с.

4.Haines, Real–Time Rendering, Fourth Edition [Текст] / Eric Haines – USA, CRC Press – 1199 с.

5.UE4 Docs [Електронний ресурс] / UE4 Docs – Режим доступу. https://docs.unrealengine.com/en–US – Загол. з екрану (дата звернення:13.03.2020).