Projekt Grafika Komputerowa

LOD – Demo

Wprowadzenie

Celem projektu jest demonstracja działania mechanizmu **Level of Detail (LOD)** w silniku Unity. LOD to technika optymalizacyjna stosowana w grafice 3D, która polega na dynamicznym zmniejszaniu szczegółowości modeli 3D w zależności od ich odległości od kamery. Dzięki temu możliwe jest znaczne zmniejszenie liczby przetwarzanych trójkątów bez zauważalnego pogorszenia jakości grafiki z perspektywy gracza.

W repozytorium przedstawiono dwa podejścia do implementacji LOD:

- z wykorzystaniem wbudowanego komponentu `LODGroup` Unity,
- za pomocą niestandardowego kontrolera `CustomLODController.cs`.

Do celów demonstracyjnych wykorzystano proste modele 3D wykonane samodzielnie w **Blenderze** – są to:

- **House_LOD0**, **House_LOD1**, **House_LOD2** – różne wersje domu o zmniejszającej się liczbie detali,
- **Car_LOD0**, **Car_LOD1**, **Car_LOD2** –samochód o różnej szczegółowości.
- **Well_LOD0**, **Well_LOD1**, **Well_LOD2** – studnia o różnej szczegółowości.

Opis działania + pliki źródłowe

2.1 Wbudowany LOD (LODGroup)

- Komponent LODGroup pozwala ustawić różne modele (mesh) lub poziomy detalu dla obiektu.
- Automatycznie przełącza modele w zależności od odległości od kamery.
- Konfiguracja odbywa się w edytorze Unity (Inspector).
- Obiekty z LODGroup można znaleźć w scenie Scenes/LOD_Demo z podpisem "LODGROUP".

  W projekcie komponent ten został przypisany do obiektów **House_LODGroup, Car_LODGroup i Well_LODGroup**, gdzie każdy poziom (LOD0, LOD1, LOD2) odwołuje się do wersji modelu o odpowiedniej złożoności.

2.2 Niestandardowy LOD (CustomLODController.cs)

- Skrypt CustomLODController.cs ręcznie zarządza poziomami detalu.
- Na podstawie odległości od kamery przełącza GameObjecty odpowiadające różnym poziomom LOD.
- Obiekty z CustomLod można znaleźć w scenie Scenes/LOD_Demo z podpisem "CUSTOMLOD"

2.3 Kody źródłowe

CustomLodController.cs - niestandardowy algorytm wyświetlania danego modelu w zależności od odległości od kamery

```
using UnityEngine;

public class CustomLODController : MonoBehaviour {
    public GameObject lod0;
    public GameObject lod1;
    public GameObject lod2;

    public float lod1Distance = 20f;
    public float lod2Distance = 40f;

    private Camera mainCamera;

    void Start() {
        mainCamera = Camera.main;
    }

    void Update() {
        float distance = Vector3.Distance(transform.position, mainCamera.transform.position);

        if (distance < lod1Distance) {
            SetLOD(0);
        }
        else if (distance < lod2Distance) {
            SetLOD(1);
        }
        else {
            SetLOD(2);
        }
    }

    void SetLOD(int level) {
        lod0.SetActive(level == 0);
        lod1.SetActive(level == 1);
        lod2.SetActive(level == 2);
    }
}
```

FreeCameraController.cs – skrypt odpowiadający za sterowanie kamerą w scenie (obracanie, przybliżanie/oddalanie, ruch kamery)

```
using UnityEngine;

public class FreeCameraController : MonoBehaviour {
    public float moveSpeed = 10f;
    public float lookSpeed = 2f;
    private float yaw = 0f;
    private float pitch = 0f;

    void Start() {
        Cursor.lockState = CursorLockMode.Locked;

        Vector3 angles = transform.eulerAngles;
        yaw = angles.y;
        pitch = angles.x;
    }

    void Update() {
        yaw += lookSpeed * Input.GetAxis("Mouse X");
        pitch -= lookSpeed * Input.GetAxis("Mouse Y");
        pitch = Mathf.Clamp(pitch, -90f, 90f);
        transform.eulerAngles = new Vector3(pitch, yaw, 0.0f);

        float horizontal = Input.GetAxis("Horizontal");
        float vertical = Input.GetAxis("Vertical");
        float upDown = 0f;

        if (Input.GetKey(KeyCode.E)) upDown -= 1f;
        if (Input.GetKey(KeyCode.Q)) upDown += 1f;

        Vector3 move = new Vector3(horizontal, upDown, vertical);
        transform.Translate(move * moveSpeed * Time.deltaTime);
    }
}
```

LODChecker.cs - wypisywanie w consoli aktualnego poziomu LOD obiektów w scenie.

```
using UnityEngine;

public class LODChecker : MonoBehaviour {
    private LODGroup lodGroup;
    private CustomLODController customLOD;
    private Camera cam;
    private Renderer rend;

    void Start() {
        lodGroup = GetComponent<LODGroup>();
        customLOD = GetComponent<CustomLODController>();
        cam = Camera.main;
        rend = GetComponentInChildren<Renderer>();
    }

    void Update() {
        if (customLOD != null) {
            int customLODLevel = GetActiveCustomLOD();
            Debug.Log(gameObject.name + " ? Custom LOD: " + customLODLevel);
        } else if (lodGroup != null) {
            float relativeHeight = GetScreenRelativeHeight();
            int lodGroupLevel = GetLODGroupIndex(relativeHeight);
            Debug.Log(gameObject.name + " ? LODGroup LOD: " + lodGroupLevel);
        } else {
            Debug.LogWarning(gameObject.name + " nie ma LODGroup ani CustomLODController!");
        }
    }

    float GetScreenRelativeHeight() {
        if (rend == null || cam == null)
            return 0;

        Bounds bounds = rend.bounds;
        Vector3 center = bounds.center;
        float distance = Vector3.Distance(cam.transform.position, center);

        float height = 2f * Mathf.Tan(0.5f * cam.fieldOfView * Mathf.Deg2Rad) * distance;
        return bounds.size.magnitude / height;
    }

    int GetLODGroupIndex(float relativeHeight) {
        LOD[] lods = lodGroup.GetLODs();

        for (int i = 0; i < lods.Length; i++) {
            if (relativeHeight >= lods[i].screenRelativeTransitionHeight)
                return i;
        }
        return lods.Length - 1;
    }

    int GetActiveCustomLOD() {
        if (customLOD.lod0 != null && customLOD.lod0.activeSelf) return 0;
        if (customLOD.lod1 != null && customLOD.lod1.activeSelf) return 1;
        if (customLOD.lod2 != null && customLOD.lod2.activeSelf) return 2;
        return -1; // Nieaktywny lub coś nie tak
    }
}
```

SceneStatsDisplay.cs - wypisywanie na ekranie liczby klatek na sekundę (FPS) i ilości obecnie wyświetlonych trójkątów.

```
using UnityEngine;

public class SceneStatsDisplay : MonoBehaviour {
    private float deltaTime = 0.0f;

    void Update() {
        deltaTime += (Time.unscaledDeltaTime - deltaTime) * 0.1f;
    }

    void OnGUI() {
        int w = Screen.width, h = Screen.height;

        GUIStyle style = new GUIStyle();
        Rect rect = new Rect(10, 10, w, h * 2 / 100);
        style.alignment = TextAnchor.UpperLeft;
        style.fontSize = h * 2 / 50;
        style.normal.textColor = Color.white;

        float fps = 1.0f / deltaTime;
        int tris = GetTotalTriangles();

        string text = $"FPS: {fps:0.} | Triangles: {tris:N0}";
        GUI.Label(rect, text, style);
    }

    int GetTotalTriangles() {
        int totalTris = 0;
        MeshFilter[] meshFilters = FindObjectsOfType<MeshFilter>();

        foreach (MeshFilter mf in meshFilters) {
            Mesh mesh = mf.sharedMesh;
            if (mesh != null && mesh.isReadable)
                totalTris += mesh.triangles.Length / 3;
        }

        SkinnedMeshRenderer[] skinned = FindObjectsOfType<SkinnedMeshRenderer>();
        foreach (SkinnedMeshRenderer smr in skinned) {
            Mesh mesh = smr.sharedMesh;
            if (mesh != null && mesh.isReadable)
                totalTris += mesh.triangles.Length / 3;
        }

        return totalTris;
    }
}
```

Uruchomienie

- Pobranie repozytorium https://github.com/napieralal/LOD-Demo
- Otwarcie projektu w Unity
- Załadowanie sceny “LOD_Demo.unity”
- Odpalenie sceny (“Play”)
- Sterowanie kamerą:
  WASD – ruch,
  Obracanie myszą - obrót kamery,
  Scroll - przybliżenie/oddalenie kamery
- Obserwacja – obiekty zmieniają model w zależności od oddalenia kamery, poziomy LOD wypisane w konsoli, na ekranie licznik FPS i trójkątów.

Wnioski

Zastosowanie techniki LOD jest jednym z kluczowych elementów optymalizacji grafiki 3D – zarówno w grach, jak i w aplikacjach wizualizacyjnych. Dzięki niej możliwe jest znaczne ograniczenie liczby renderowanych detali tam, gdzie nie są one istotne dla użytkownika (np. w tle sceny).

Nawet prosta implementacja, taka jak ta przedstawiona w projekcie, pozwala zauważalnie poprawić wydajność przy dużej liczbie obiektów w scenie. Samodzielne przygotowanie modeli w Blenderze w różnych poziomach szczegółowości pozwala precyzyjnie kontrolować, jak wiele detali zostaje usuniętych na każdym etapie.

Technika LOD:

- poprawia **wydajność** bez dużej straty jakości wizualnej,
- jest łatwa do zaimplementowania w silniku Unity zarówno poprzez gotowe komponenty (`LODGroup`), jak i własne skrypty,
- dobrze współgra z innymi metodami optymalizacji (culling, batching, occlusion).

Warto rozważać jej stosowanie nie tylko w przypadku rozbudowanych scen, ale także w mniejszych projektach, które mają być skalowalne lub dostępne na słabszych urządzeniach.