Enhancement Two: Algorithms and Data Structure

Rafael V. Canseco

SNHU

CS-499 Computer Science Capstone

Artifact Overview

The artifact selected for this milestone is the *SceneManager* component from my CS-330 Computational Graphics and Visualization project. The original artifact was developed in early 2024 as part of a 3D rendering assignment involving shaders, texture loading, lighting, and transformation pipelines. The *SceneManager.cpp* file is responsible for loading textures, managing materials, setting shader parameters, and drawing 3D primitives that make up the rendered scene.

Although the original artifact functioned correctly, it contained several inefficiencies and structural limitations, particularly in regard to linear-time lookups, repetitive code, and inconsistent texture-management behavior. These limitations made the artifact an ideal candidate for enhancement in the area of algorithms and data structures.

Justification for Inclusion in the ePortfolio

This artifact was selected because it demonstrates my ability to apply core algorithmic principles and effectively use data structures—foundational outcomes of the Computer Science program. The enhancements significantly improved both time complexity and maintainability.

O(1) Lookup Tables Using Hash Maps

I introduced three *unordered_map* structures into the SceneManager:

- **m_textureIdLookup** — maps texture tags to OpenGL texture IDs
- **m_textureSlotLookup** — maps texture tags to GPU texture-unit indices
- **m_materialLookup** — maps material tags to complete material structures

These hash maps replaced the original linear-search loops, which scanned up to sixteen entries for every lookup. The previous design resulted in O(n) operations, whereas the enhanced version performs lookups in O(1) average time. This demonstrates an ability to evaluate and select appropriate data structures to optimize computational cost.

Expanded Material Lookup Optimization

In addition to texture optimization, I implemented a fast-access hash map for material data. Previously, the program performed a linear search through a vector of materials each time a

material was referenced. By converting this process into hash-based retrieval, material access now benefits from the same performance improvements as texture and slot lookups. This change also improves architectural consistency within the SceneManager.

Performance Benefits in Real-Time Rendering

Rendering functions execute once per frame and, in many cases, multiple times per object. Because of this, even small inefficiencies have cumulative performance impacts. By removing linear searches from the rendering loop, CPU overhead was reduced substantially across frames.

The shift from O(n) to O(1) operations provides measurable improvements in rendering efficiency. This is particularly important in graphics systems where every millisecond affects visual smoothness and responsiveness.

Algorithmic Design Trade-Offs

Selecting hash maps introduced intentional design considerations. Hash-based structures require:

- Increased memory overhead compared to arrays
- Dependence on hash distribution quality
- Initial construction time during scene preparation

However, the performance benefits in repeated lookups far outweigh these trade-offs in a real-time rendering context. Evaluating these complexities reflects an understanding of how computational theory applies to practical software engineering challenges.

Outcome Coverage Reflection

In Module One, I identified that this enhancement would demonstrate progress toward the following program outcome:

"Design and evaluate computing solutions using algorithmic principles and computer science practices and standards appropriate to its solution."

This outcome was fully met. By reducing lookup operations from O(n) to O(1), I significantly improved computational efficiency. This enhancement required analyzing the original algorithmic structure, assessing trade-offs, and applying modern data structures effectively.

The work also supports the program outcome:

"Demonstrate an ability to use innovative techniques, skills, and tools in computing practices for the purpose of implementing computer solutions."

Correcting resource lifecycle handling, such as replacing incorrect OpenGL deletion methods, and refactoring repeated rendering code demonstrate applied innovation and responsible modernization of an existing codebase

Reflection on the Enhancement Process

Enhancing this artifact deepened my understanding of how data structures influence performance in real-time graphics environments. Several key insights were gained throughout the process:

1. **Importance of Data Structures**
   Small inefficiencies within per-frame operations accumulate across rendering cycles. Implementing hash-map lookups provided substantial performance improvements.
2. **Debugging Legacy Code**
   Correcting an incorrect OpenGL texture-deletion method emphasized the importance of understanding GPU memory management.

3. **Improved Maintainability**
   Refactoring repeated block sections into reusable helper functions enhanced readability and reduced future maintenance costs.
4. **Backward Compatibility**
   I preserved the original linear-search implementation as a fallback. This demonstrated responsible enhancement practices by ensuring legacy behavior remained intact if needed.

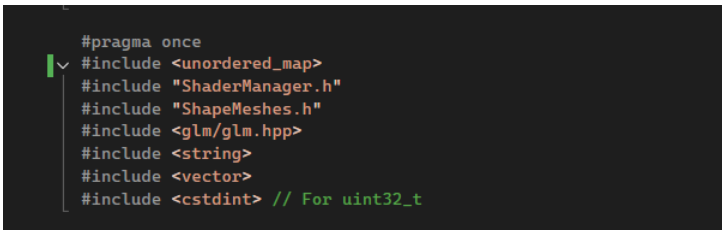
Additionally, I learned the importance of keeping lookup structures synchronized with underlying vectors to prevent stale references or mismatched data. Maintaining this consistency is crucial for a stable rendering pipeline.

Conclusion

This enhancement improved the artifact's efficiency, readability, and overall performance. The improved SceneManager now demonstrates stronger architectural design, optimized data access, and a more modern approach to resource handling. These improvements reflect my progression as a computer scientist and make the artifact a valuable component of my final ePortfolio.

Here are snippet pictures of the before and after enhancement:

SceneManager.h before enhancement:



```
#pragma once
#include <unordered_map>
#include "ShaderManager.h"
#include "ShapeMeshes.h"
#include <glm/glm.hpp>
#include <string>
#include <vector>
#include <cstdint> // For uint32_t
```

```
    // Pointer to shader manager object
    ShaderManager* m_pShaderManager;
    // Pointer to basic shapes object
    ShapeMeshes* m_basicMeshes;
    // Total number of loaded textures
    int m_loadedTextures;
    // Loaded textures info
    TEXTURE_INFO m_textureIDs[16];
    // Defined object materials
    std::vector<OBJECT_MATERIAL> m_objectMaterials;

    // Load texture images and convert to OpenGL texture data
    bool CreateGLTexture(const char* filename, std::string tag);
    // Bind loaded OpenGL textures to slots in memory
    void BindGLTextures();
    // Free the loaded OpenGL textures
    void DestroyGLTextures();
    // Find a loaded texture by tag
    int FindTextureID(std::string tag);
    int FindTextureSlot(std::string tag);
    // Find a defined material by tag
    bool FindMaterial(std::string tag, OBJECT_MATERIAL& material);
```

SceneManager.h after enhancement:

```
#include <unordered_map>
#include "ShaderManager.h"
#include "ShapeMeshes.h"
#include <glm/glm.hpp>
#include <string>
#include <vector>
#include <cstdint> // For uint32_t

/***********************************************************
*  SceneManager
*
*  This class contains the code for preparing and rendering
*  3D scenes, including the shader settings.
***********************************************************/
class SceneManager
{
public:
    // Constructor
    SceneManager(ShaderManager* pShaderManager);
    // Destructor
    ~SceneManager();

    struct TEXTURE_INFO
    {
        std::string tag;
        uint32_t ID;
    };

    struct OBJECT_MATERIAL
    {
        glm::vec3 diffuseColor;
        glm::vec3 specularColor;
        float shininess;
        std::string tag;
    };

private:
    // Pointer to shader manager object
    ShaderManager* m_pShaderManager;
    // Pointer to basic shapes object
    ShapeMeshes* m_basicMeshes;
    // Total number of loaded textures
    int m_loadedTextures;
    // Loaded textures info
    TEXTURE_INFO m_textureIDs[16];
    // Defined object materials
    std::vector<OBJECT_MATERIAL> m_objectMaterials;

    // Fast lookup for texture and materias (enhancement for CS-499)
    std::unordered_map<std::string, int>    m_textureSlotLookup; // tag -> texture slot
    std::unordered_map<std::string, GLuint>     m_textureIdLookup; // tag -> OpengL texture ID
    std::unordered_map<std::string, OBJECT_MATERIAL>    m_materialLookup; // tag -> material
```

SceneManager.cpp before enhancement:

```
        // Pointer to shader manager object
        ShaderManager* m_pShaderManager;
        // Pointer to basic shapes object
        ShapeMeshes* m_basicMeshes;
        // Total number of loaded textures
        int m_loadedTextures;
        // Loaded textures info
        TEXTURE_INFO m_textureIDs[16];
        // Defined object materials
        std::vector<OBJECT_MATERIAL> m_objectMaterials;

        // Load texture images and convert to OpenGL texture data
        bool CreateGLTexture(const char* filename, std::string tag);
        // Bind loaded OpenGL textures to slots in memory
        void BindGLTextures();
        // Free the loaded OpenGL textures
        void DestroyGLTextures();
        // Find a loaded texture by tag
        int FindTextureID(std::string tag);
        int FindTextureSlot(std::string tag);
        // Find a defined material by tag
        bool FindMaterial(std::string tag, OBJECT_MATERIAL& material);
```

```
/***********************************************************
 *  FindTextureID()
 *
 *  This method is used for getting an ID for the previously
 *  loaded texture bitmap associated with the passed in tag.
 ************************************************************/
int SceneManager::FindTextureID(std::string tag)
{
    int textureID = -1;
    int index = 0;
    bool bFound = false;

    while ((index < m_loadedTextures) && (bFound == false))
    {
        if (m_textureIDs[index].tag.compare(tag) == 0)
        {
            textureID = m_textureIDs[index].ID;
            bFound = true;
        }
        else
            index++;
    }

    return(textureID);
}
```

```
/***********************************************************
 *  FindTextureSlot()
 *
 *  This method is used for getting a slot index for the previously
 *  loaded texture bitmap associated with the passed in tag.
 ************************************************************/
int SceneManager::FindTextureSlot(std::string tag)
{
    int textureSlot = -1;
    int index = 0;
    bool bFound = false;

    while ((index < m_loadedTextures) && (bFound == false))
    {
        if (m_textureIDs[index].tag.compare(tag) == 0)
        {
            textureSlot = index;
            bFound = true;
        }
        else
            index++;
    }

    return(textureSlot);
}
```

```
            m_objectMaterials.push_back(softMaterial); // Add material to the list

            // Clay Material
            OBJECT_MATERIAL clayMaterial;
            clayMaterial.diffuseColor = glm::vec3(0.8f, 0.5f, 0.3f); // Terracotta clay color
            clayMaterial.specularColor = glm::vec3(0.2f, 0.1f, 0.05f); // Subtle reflectivity for clay
            clayMaterial.shininess = 16.0f; // Slight shine to simulate ceramic finish
            clayMaterial.tag = "clay"; // Tag to identify clay material in the scene

            m_objectMaterials.push_back(clayMaterial); // Add material to the list

        }


    /***********************************************************
     *  FindMaterial()
     *
     *  This method is used for getting a material from the previously
     *  defined materials list that is associated with the passed in tag.
     ***********************************************************/
    bool SceneManager::FindMaterial(std::string tag, OBJECT_MATERIAL& material)
    {
        if (m_objectMaterials.size() == 0)
        {
            return(false);
        }

        int index = 0;
        bool bFound = false;
        while ((index < m_objectMaterials.size()) && (bFound == false))
        {
            if (m_objectMaterials[index].tag.compare(tag) == 0)
            {
                bFound = true;
                material.diffuseColor = m_objectMaterials[index].diffuseColor;
                material.specularColor = m_objectMaterials[index].specularColor;
                material.shininess = m_objectMaterials[index].shininess;
            }
            else
            {
                index++;
            }
        }

        return(true);
    }
```

SceneManager.cpp after:

```
        // Cs-499 Enhancement: maintain fast lookup tables for textures
        int slotIndex = m_loadedTextures;   // slot we just used
        m_textureIdLookup[tag] = textureID;     //tag -> GL texture ID
        m_textureSlotLookup[tag] = slotIndex;      //tag -> texture slot

        // Now that everything is registered, advance the count
        m_loadedTextures++;

        return true;


    }

    std::cout << "Could not load image:" << filename << std::endl;

    // Error loading the image
    return false;
}
```

```cpp
int SceneManager::FindTextureID(std::string tag) {

    // CS-499 Enhancement: 0(1) average lookup using an unordered_map
    auto it = m_textureIdLookup.find(tag);
    if (it != m_textureIdLookup.end()) {
        return static_cast<int>(it->second);

    }
    // Fallback to legacy method if not found in cache
    int textureID = -1;
    int index = 0;
    bool bFound = false;

    while ((index < m_loadedTextures) && (bFound == false)) {
        if (m_textureIDs[index].tag.compare(tag) == 0) {
            textureID = m_textureIDs[index].ID;
            bFound = true;

        }
        else {
            index++;

        }
    }
    return textureID;

}
```

```cpp
int SceneManager::FindTextureSlot(std::string tag) {
    // CS-499 Enhancement: 0(1) average lookup using an unordered_map

    auto it = m_textureSlotLookup.find(tag);
    if (it != m_textureSlotLookup.end()) {
        return it->second;

    }

    // Fallback to legacy linear search if not found in cache

    int textureSlot = -1;
    int index = 0;
    bool bFound = false;

    while ((index < m_loadedTextures) && (bFound == false)) {
        if (m_textureIDs[index].tag.compare(tag) == 0) {
            textureSlot = index;
            bFound = true;

        }
        else {
            index++;

        }
    }
    return textureSlot;

}
```

```cpp
bool SceneManager::FindMaterial(std::string tag, OBJECT_MATERIAL& material) {

    // CS-499 Enhancement: O(1) average lookup using a hash map
    auto it = m_materialLookup.find(tag);
    if (it != m_materialLookup.end()) {
        material = it->second;
        return true;
    }

    // Fallback to legacy linear search if map is not populated
    if (m_objectMaterials.empty()) {
        return false;

    }
    int index = 0;
    bool bFound = false;
    while ((index < static_cast<int>(m_objectMaterials.size())) && (bFound == false)) {
        if (m_objectMaterials[index].tag.compare(tag) == 0) {
            bFound = true;
            material = m_objectMaterials[index];

        }
        else {
            index++;

        }
    }
    return bFound;
}
```

```cpp
        clayMaterial.tag = "clay";
        m_objectMaterials.push_back(clayMaterial);

        // CS-499 Enhancement: build fast lookup map for materials
        m_materialLookup.clear();
        for (const auto& mat : m_objectMaterials) {
            m_materialLookup[mat.tag] = mat;
        }
    }
```

Functional Code after Enhancement