Ericka Resendez

Southern New Hampshire University

CS-499 Computer Science Capstone

Artifact Narrative: 3D Replication Scene

1. Briefly describe the artifact. What is it? When was it created?
    a. This artifact is a 3D interactive scene that was developed for CS-330 Computational Graphics and Visualization course during November of 2024. The original project was a static 3D scene depicting Bluey's backyard, featuring a tree with branches, an octagonal bench, bushes, and flowers rendered using OpenGL primitives like cylinders, spheres, prisms, and tori. Users could navigate the scene using keyboard (WASD for movement, QE for vertical movement, OP for perspective/orthographic view switching) and mouse controls, but the camera could freely pass through all objects without physical boundaries.
    b. The enhanced version transforms this into a physically realistic environment by implementing a comprehensive collision detection system. The system includes three bounding volume types defined in custom data structures: BoundingSphere, BoundingBox, and BoundingCapsule. The collision system enables realistic camera movement where users cannot walk through solid objects, creating an immersive first-person navigation experience with physical boundaries that match the visual scene geometry.
2. Justify the inclusion of the artifact in your ePortfolio. Why did you select this item? What specific components of the artifact showcase your skills and abilities in software

development? How did the enhancement improve the artifact? What specific skills did you demonstrate in the enhancement?

a. I selected this artifact because it demonstrates my ability to implement complex mathematical algorithms and integrate them seamlessly into an existing codebase. The enhancement showcases several critical software development competencies through concrete implementations. For algorithm design and computational geometry, I implemented three distinct collision detection algorithms in SceneManager.cpp: SphereSphereCollision() calculates the distance between two sphere centers and compares it against the sum of their radii using glm::distance(); SphereBoxCollision() finds the closest point on an axis-aligned bounding box to a sphere center using glm::clamp() to constrain each coordinate between the box's minimum and maximum values, then checks if that closest point falls within the sphere's radius; and SphereCapsuleCollision() uses PointToLineDistance() to project the sphere's center onto the line segment between the capsule's base and tip points, clamping the projection parameter between 0 and 1 to handle endpoints, demonstrating understanding of vector mathematics and line-point distance calculations. For system architecture and integration, I designed the collision system in three logical layers: InitializeCollisionObjects() populates collision volumes for all scene objects by analyzing their visual geometry and creating matching bounding volumes (for example, the tree trunk at position (-3.0f, 0.0f, -3.0f) to (2.5f, 14.0f, 2.5f) becomes a BoundingBox, while the main tree foliage sphere centered at (0.0f, 18.0f, 0.0f) with 4.5f radius becomes a BoundingSphere); CheckCameraCollision() tests the proposed camera position

against all collision volumes by iterating through the three vectors (m_sphereColliders, m_boxColliders, m_capsuleColliders) and calling the appropriate collision detection method, returning true on the first collision found for efficiency; and ResolveCollision() implements sliding collision response by attempting incremental movement in 10% steps toward the desired position until finding a collision-free location, allowing smooth movement along obstacle surfaces rather than complete blocking. For real-world integration challenges, I solved the circular dependency problem between ViewManager (which handles camera movement) and SceneManager (which performs collision checks) by using forward declarations in headers and passing a SceneManager pointer to ViewManager's constructor, stored as m_pSceneManager, enabling ViewManager::ProcessKeyboardEvents() to call m_pSceneManager->ResolveCollision() during movement processing. I also implemented a debug visualization system using DebugDrawCollisionBounds() that renders collision volumes as wireframe overlays by temporarily disabling depth testing with glDisable(GL_DEPTH_TEST), switching to line rendering mode with glPolygonMode(GL_FRONT_AND_BACK, GL_LINE), drawing color-coded shapes (red for spheres, yellow for capsules, green for boxes), and critically restoring solid rendering mode with glPolygonMode(GL_FRONT_AND_BACK, GL_FILL) to prevent all subsequent scene geometry from rendering as wireframes—this was initially a major bug that took careful state management to resolve. The enhancement transformed the artifact from a passive 3D visualization into an interactive navigable environment with realistic physical

constraints, demonstrating practical application of computational geometry in game development and simulation contexts where spatial awareness and collision prevention are essential for user experience.

b.

3. Reflect on the process of enhancing the artifact. What did you learn as you were creating it and improving it? What challenges did you face? How did you incorporate feedback as you made changes to the artifact? How was the artifact improved? Which course outcomes did you partially or fully meet with your enhancements? Which do you feel were not met?

    a. This enhancement deepened my understanding of computational geometry and real-time spatial reasoning algorithms. I learned that collision detection requires careful consideration of both accuracy and performance, so I checked the camera against static scene geometry and implemented early-exit logic in CheckCameraCollision() that returns true immediately upon finding the first collision rather than continuing to test remaining objects. I also added the debug visualization toggle using the C key to provide visual confirmation that collisions, enabling developers and testers to verify collision volume placement without modifying code.

    b. The most significant technical challenges involved OpenGL state management and architectural integration. The debug visualization bug was challenging as after implementing DebugDrawCollisionBounds(), the entire scene rendered as wireframes instead of solid objects. I discovered that glPolygonMode(GL_FRONT_AND_BACK, GL_LINE) persists as a global

OpenGL state until explicitly reset, so any rendering code after the debug function would use line mode. The fix required saving the current polygon mode before drawing wireframes, then restoring it immediately after to prevent wireframe mode from affecting all subsequent rendering.

c. I did not incorporate feedback from my professor as I was unable to submit a version in time. The collision system was improved by adjusting bounding volumes to match the visual shapes, adding smooth sliding responses, and creating a debug view to visualize the collision boundaries.

d. I fully met course outcome three through this enhancement by designing and evaluating multiple collision detection algorithms with different computational complexities and accuracy tradeoffs, and applying vector mathematics and geometric algorithms to solve spatial reasoning problems. I also fully met course outcome four by delivering a professional-quality collision system using industry-standard techniques (bounding volumes, spatial partitioning concepts), integrating third-party libraries (GLM for vector mathematics) appropriately, and implementing comprehensive debug visualization tools that would be valuable in production game development. Course outcome five was not addressed by this enhancement, as collision detection focuses on spatial simulation rather than security concerns like input validation or access control.