Bazy danych 2 - dokumentacja projektu

Obsługa danych przestrzennych trójwymiarowych

25.05.24r., Aleksandra Śliwska

Spis treści

1 Opis problemu i rozwiązania - funkcjonalności API

Tematem projektu była „Obsługa danych przestrzennych trójwymiarowych". W ramach zadania należało przygotować API i jego implementację w postaci aplikacji umożliwiającej przetwarzanie danych trójwymiarowych z wykorzystaniem typów danych CLR UDT oraz metod CLR UDF. Przygotowane metody miały umożliwić wyznaczenie odległości pomiędzy punktami, wyznaczenie pola określonej powierzchni, objętości obszaru zamkniętego oraz sprawdzenie czy punkt należy do danego obszaru zamkniętego lub powierzchni.

W celu wykonania zadania napisano skrypt inicjalizujący bazę danych SQL Server z użyciem bibliotek DLL. Biblioteki te zawierają skrypty CLR - User Defined Types, User Defined Functions, User Defined Aggregates - napisane w C#. Zdefiniowano także tabele, funkcje, wyzwalacze i procedury w T-SQL.

Na takiej bazie danych operuje TriangleMeshAPI napisane w C# i łączące się z serwerem (lokalnym lub w chmurze) dzięki ADO.NET. Od użytkownika potrzebuje jedynie connection-string'a do połączenia z bazą. W zamian udostępnia 3 własne typy danych (punkt, trójkąt, mesh) opakowujące informacje z bazy do ich prostego użytkowania oraz szereg metod operujących na bazie lub pobierających i konsolidujących z niej informacje.

Do przetestowania API napisano szereg testów integracyjnych w C# z użyciem NUnit oraz Fluent Assertions. Testują one pojedyncze funkcjonalności realizowane przez API i bazę. Testy działają

szybko dla baz lokalnych i testują rzeczywiste zachowanie API (ponieważ wszystkie publiczne metody API otwierają połączenie z bazą danych).

Działanie API przedstawiono przez implementację jego funkcjonalności w aplikacji napisanej w silniku graficznym Unity. Pozwala ona na poruszanie się w przestrzeni 3D z narysowanymi osiami XYZ, renderowanie informacji zawartych w bazie, zaznaczanie odpowiednich struktur (punkt, trójkąt, mesh) i wywoływanie dla nich funkcji z API przez klikanie odpowiednich przycisków w UI.

2 Opis typów danych oraz metod udostępnianych przez API

TriangleMeshAPI udostępnia 4 typy danych:

1. `MeshObject` - struktura, po której dziedziczą wszystkie inne udostępniane typy danych (`Point`, `Triangle` i `Mesh`) - posiada:
   - `int id` - `id` obiektu w bazie danych
   - konstruktor `MeshObject(int id)` - ustawiający `id`
   - konstruktor pusty - ustawiający `id` na -1
2. `Point` - reprezentujący punkt w przestrzeni 3D - posiada:
   - `double x, y, z` - koordynaty punktu w przestrzeni 3D
   - konstruktor `Point(int id, double x, double y, double z)` - ustawiający zmienne `id, x, y, z`
   - konstruktor pusty - ustawiający `x, y, z` na 0
   - `string ToString()` - metoda zmieniająca `Point` na tekst, np.: „(p 4: [3.5, 2, 0.664])", gdzie pierwsza liczba to `id`, a kolejne to koordynaty.
3. `Triangle` - reprezentujący trójkąt złożony z 3 punktów - posiada:
   - `Point a, b, c` - referencje do 3 obiektów `Point`, które są jego wierzchołkami
   - konstruktor `Triangle(int id, Point a, Point b, Point c)` - ustawiający zmienne `id, a, b, c`
   - konstruktor pusty - ustawiający `a, b, c` na null
   - `string ToString()` - metoda zmieniająca `Triangle` na tekst, np.: „(t 4: <a>, <b>, <c>)", gdzie pierwsza liczba to `id`, z kolei <a>, <b>, <c> to wywołane metody `Point.ToString()` na kazdym z punktów.
4. `Mesh` - reprezentujący mesh (strukturę 3D) złożoną z trójkątów - posiada:
   - `List<Triangle> triangles` - listę referencji do obiektów `Triangle`, które tworzą dany mesh
   - konstruktor `Mesh(int id, List<Triangle> triangles)` - ustawiający zmienne `id`, `triangles`
   - konstruktor pusty - ustawiający `triangles` na null
   - `string ToString()` - metoda zmieniająca `Mesh` na tekst, np.: „(m 4:<list>)", gdzie pierwsza liczba to `id`, z kolei <list> to wywołane metody `Triangle.ToString()` na kazdym `Triangle`, połączone przecinkami i drukowane w nowych linijkach, poprzedzonych znakiem tabulacji.

Struktury te wykorzystane są w implementacji poniższych metod (oprócz konstruktora wszystkie otwierają połączenie z bazą danych).

1. `TriangleMeshAPI(string connectionString)` - konstruktor ustawiający ADO.NET connection string do łączenia się z bazą danych.

2. `List<Point> fetchPoints()` - pobiera informacje i punktach z bazy danych (id, x, y, z) i zwraca nowo utworzoną z nich listę obiektów `Point`.
3. `List<Triangle> fetchTriangles(List<Point> points)` - pobiera informacje o trójkątach z bazy danych (triangle id, punkt a id, punkt b id, punkt c id) i zwraca nowo utworzoną z nich listę obiektów `Triangle`, dodając do nich referencje do odpowiednich obiektów `Point` przekazanych jako parametr metody. Wyrzuca `ArgumentException`, jeżeli w liście punktów zabrakło punktu wykorzystywanego przez trójkąt lub jeżeli id punktu w liście się powtórzyło.
4. `List<Mesh> fetchMeshes(List<Triangle> triangles)` - pobiera informacje o meshach z bazy danych (mesh id, lista trójkątów) i zwraca nowo utworzoną z nich listę obiektów `Mesh`, dodając do nich referencje do odpowiednich obiektów `Triangle` przekazanych jako parametr metody. Wyrzuca `ArgumentException`, jeżeli w liście trójkątów zabrakło trójkąta wykorzystywanego przez mesh lub jeżeli id trójkąta w liście się powtórzyło.
5. `int insertPoint(double x, double y, double z)` - tworzy nowy punkt o podanych koordynatach w bazie danych i zwraca id obiektu w bazie danych.
6. `int insertTriangle(Point a, Point b, Point c)` - tworzy nowy trójkąt o podanych wierzchołkach w bazie danych i zwraca id obiektu w bazie danych. Id punktów są sortowane rosnąco przed wstawieniem. Wyrzuca `SqlException` jeżeli punkt o podanych wierzchołkach już istnieje w bazie danych, lub id punktów w trójkącie się powtórzyły, lub punkty o podanych id nie istnieją w bazie danych.
7. `int insertTriangle(int aId, int bId, int cId)` - to samo co funkcja `int insertTriangle(Point a, Point b, Point c)`, jedynie pobiera od razu id punktów wierzchołkowych.
8. `int insertMesh(List<Triangle> triangles)` - tworzy nowy mesh o podanych trójkątach w bazie danych i zwraca id obiektu w bazie danych. Wyrzuca `SqlException` jeżeli id trójkątów się powtarzają.
9. `int insertMesh(List<int> triangleIds)` - to samo co funkcja `int insertMesh(List<Triangle> triangles)`, jedynie pobiera od razu id trójkątów.
10. `List<Mesh> mergeMeshes(List<int> meshIds, List<Triangle> allTriangles)` - tworzy nowy mesh w bazie danych, przypisuje mu trójkąty wszystkich meshy z listy `meshIds` oraz usuwa meshe na tej liście z bazy danych. Następnie zwraca aktualną listę meshy w bazie danych uzupełnioną referencjami do trójkątów z `allTriangles`. Nie zmienia nic w bazie, jeżeli podane zostanie 1 lub 0 id meshy do połączenia. Wyrzuca `ArgumentException`, jeżeli na liście `allTriangles` brakuje trójkąta wykorzystywanego w jakimkolwiek meshu w bazie.
11. `void deletePoints(List<int> pointIds, out List<Point> points, out List<Triangle> triangles, out List<Mesh> meshes)` - usuwa punkty o id na liście `pointIds` z bazy danych i zwraca aktualne listy danych `points, triangles` i `meshes`. Usunięcie punktu powoduje usunięcie wszystkich trójkątów, które mają w nim wierzchołek. Jeżeli usunięte zostaną wszystkie trójkąty należące do danego meshu, on też jest usuwany. Usunięcie wierzchołka o największym id powoduje, że indeksacja nowych punktów rozpoczyna się od największego istniejącego id + 1.
12. `void deleteTriangles(List<int> triangleIds, List<Point> points, out List<Triangle> triangles, out List<Mesh> meshes)` - usuwa trójkąty o id na liście `triangleIds` z bazy danych i zwraca aktualne listy danych `triangles` i `meshes`, wypełnionych referencjami do punktów z listy `points`. Jeżeli usunięte zostaną wszystkie trójkąty należące do danego meshu, on też jest usuwany. Usunięcie trójkąta o największym id powoduje, że indeksacja nowych trójkątów rozpoczyna się od największego istniejącego id + 1. Wyrzuca `ArgumentException`, jeżeli na liście `points` brakuje punktu wykorzystywanego w jakimkolwiek zwracanym trójkącie na liście `triangles`.
13. `void deleteMeshes(List<int> meshIds)` - usuwa meshe o id na liście `meshIds` z bazy danych. Usunięcie meshu o największym id powoduje, że indeksacja nowych meshy rozpoczyna

się od największego istniejącego id + 1.

14. `double getDistanceBetweenPoints(int aId, int bId)` - zwraca dystans pomiędzy wierzchołkami o podanych id. Zwraca 0 dla tego samego punktu. Wyrzuca `InvalidCastException`, jeżeli punkt o podanym id nie istnieje w bazie danych.
15. `double getTriangleArea(Triangle triangle)` - zwraca pole podanego trójkąta. Zwraca 0, jeżeli trójkąt jest linią. Wyrzuca `InvalidCastException`, jeżeli punkt wierzchołkowy trójkąta o podanym id nie istnieje w bazie danych.
16. `double getMeshArea(int id)` - zwraca pole podanego meshu (sumę pól jego trójkątów). Zwraca 0, jeżeli mesh nie posiada trójkątów. Wyrzuca `InvalidCastException`, jeżeli mesh o podanym id nie istnieje w bazie danych.
17. `bool isMeshManifold(int id)` - zwraca true, jeżeli mesh o podanym id jest manifold, false w przeciwnym wypadku. Manifold mesh to siatka trójkątów, w której każda krawędź sąsiaduje z dokładnie dwoma trójkątami (czyli jest zamknięta i nie ma dodatkowych, wystających ścianek), a sam mesh jest spójny - nie da się go rozdzielić na 2 lub więcej manifold meshe. W manifold meshu nie mogą się także pojawiać „klepsydry" - które istniałyby na przykład wtedy, gdyby 2 manifold meshe połączyć jednym wierzchołkiem. Zwraca false dla pustych meshy. Wyrzuca `ArgumentException`, jeżeli mesh o podanym id nie istnieje w bazie danych.
18. `double getMeshVolume(int id)` - zwraca objętość meshu o podanym id w bazie danych. Zwraca 0 dla płaskich meshy. Wyrzuca `ArgumentException`, jeżeli mesh o podanym id nie istnieje w bazie danych albo podany mesh nie jest manifold.
19. `bool isPointInsideMesh(int pointId, int meshId)` - zwraca true, jeżeli punkt o id = `pointId` znajduje się wewnątrz meshu o id = `meshId` w bazie danych, false w przeciwnym wypadku. Warunek jest sprawdzany kilka razy dla różnych punktów zaczepienia, ale nie ma gwarancji, że w ekstremalnych przypadkach wynik otrzymany w procesie głosowania będzie prawdziwy. Zachowanie jest niezdefiniowane, jeżeli punkt leży na krawędzi lub wierzchołku meshu. Wyrzuca `ArgumentException`, jeżeli punkt albo mesh o podanym id nie istnieje w bazie albo mesh nie jest manifold.
20. `bool isPointOnTriangle(int pointId, int triangleId)` - zwraca true, jeżeli punkt o id = `pointId` znajduje się na powierzchni, bokach lub wierzchołkach trójkąta o id = `triangleId` w bazie danych, false w przeciwnym wypadku. Wyrzuca `ArgumentException`, jeżeli punkt albo trójkąt o podanym id nie istnieje w bazie.
21. `bool isPointOnMesh(int pointId, int meshId)` - zwraca true, jeżeli punkt o id = `pointId` znajduje się na powierzchni, bokach lub wierzchołkach trójkątów należących do meshu o id = `meshId` w bazie danych, false w przeciwnym wypadku. Wyrzuca `ArgumentException`, jeżeli punkt albo mesh o podanym id nie istnieje w bazie.

3 Opis implementacji API

Każda z publicznych metod w TriangleMeshAPI otwiera połączenie z bazą danych dzięki connection stringowi przekazanemu do konstruktora klasy. Metody te wysyłają proste komendy SELECT, INSERT, DELETE do bazy, aby manipulować danymi w tabelach. Mogą także wywoływać funkcje oraz procedury. Większość logiki zapytań wykonuje się właśnie w tych wbudowanych procedurach, funkcjach CLR UDF oraz dzięki wyzwalaczom i CONSTRAINT'om w tabelach.

Skrypt inicjalizujący bazę danych `[TriangleMeshAPI]` tworzy tą bazę i wszystkie potrzebne elementy wewnątrz niej.

3.1 User Defined Types

1. `Point3D` - nullable typ CLR UDT, który przechowuje koordynaty double x, y, z punktu w przestrzeni 3D. Może także reprezentować wektor w przestrzeni 3D. Posiada funkcje:

- `string ToString()` - zwracający koordynaty punktu w formie tekstowej oddzielone przecinkami, np.: „3.5,2,6.6665”
- konstruktor `Point3D(double x, double y, double z)` - tworzący punkt o koordynatach x,y,z
- konstruktor `Point3D(bool nothing)` - tworzący null point
- gettery i settery `X, Y, Z` - ustawiające i zwracające koordynaty punktu
- `Point3D Parse(SqlString s)` - parsująca koordynaty z tekstu i zwracająca nowy `Point3D`. Znakiem oddzielającym część ułamkową jest kropka.
- `Point3D getVectorTo(Point3D other)` - zwraca wektor od tego punktu do `other`.
- `bool equals(Point3D other)` - zwraca true, jeżeli koordynaty tego punktu i `other` są takie same.

3.2 User Defined Aggregates

1. `MinX` - agregat CLR UDA, operuje na typie `Point3D`. Zwraca najmniejszą wartość koordynatu x wśród wszystkich podanych mu `Point3D`.
2. `MinY` - agregat CLR UDA, operuje na typie `Point3D`. Zwraca najmniejszą wartość koordynatu y wśród wszystkich podanych mu `Point3D`.
3. `MinZ` - agregat CLR UDA, operuje na typie `Point3D`. Zwraca najmniejszą wartość koordynatu z wśród wszystkich podanych mu `Point3D`.

3.3 User Defined Functions

Poniższe funkcje są zdefiniowane jako CLR UDF (bez read access). W C# mają identyczne nazwy, jedynie z dużych liter. Wszystkie zwracają NULL on NULL input.

1. `[getDistance] (@a [Point3D], @b [Point3D])` - zwraca odległość między punktami a i b (float) ze wzoru: $||\vec{AB}||$. Wykorzystywana w API w:
   - `double getDistanceBetweenPoints(int aId, int bId)`
2. `[getTriangleArea] (@a [Point3D], @b [Point3D], @c [Point3D])` - zwraca powierzchnię trójkąta (float) ze wzoru: $\frac{||\vec{AB}\times\vec{AC}||}{2}$. Wykorzystywana w API w:
   - `double getTriangleArea(Triangle triangle)`
   - w procedurze `[getMeshArea] (@meshId int)`
3. `[checkIfPointInTriangle] (@p [Point3D], @a [Point3D], @b [Point3D], @c [Point3D])` - zwraca bit true, jeżeli punkt p znajduje się na powierzchni trójkąta a, b, c. Muszą być spełnione 2 warunki:
   - punkt musi leżeć na tej samej powierzchni: $\frac{|(\vec{AB}\times\vec{AC})\cdot\vec{AP}|}{||\vec{AB}\times\vec{AC}||} \leq 0.0001$
   - projekcja punktu na powierzchnię trójkąta (barycentryczne koordynaty) musi leżeć wewnątrz trójkąta. Muszą być spełnione:
     - $0 \leq \gamma = \frac{(\vec{AB}\times\vec{AP})\cdot(\vec{AB}\times\vec{AC})}{||\vec{AB}\times\vec{AC}||^2} \leq 1$
     - $0 \leq \beta = \frac{(\vec{AP}\times\vec{AC})\cdot(\vec{AB}\times\vec{AC})}{||\vec{AB}\times\vec{AC}||^2} \leq 1$
     - $0 \leq= 1 - \gamma - \beta \leq 1$

   W przeciwnym razie zwraca false. Wykorzystywana w API w:
   - w procedurze `[isPointOnMesh] (@pointId int, @meshId int)`

- w procedurze `[isPointOnTriangle] (@pointId int, @triangleId int)`

4. `[doesRayIntersect] (@a [Point3D], @b [Point3D], @c [Point3D], @r1 [Point3D], @r2 [Point3D])` - zwraca (bit) true, jeżeli odcinek od r1 do r2 przecina trójkąt a, b, c. Muszą być spełnione warunki:
   - znaki objętości tetrahedronów (a,b,c,r1) i (a,b,c,r2) muszą być inne
   - znaki objętości tetrahedronów (a,b,r1,r2), (b,c,r1,r2), (c,a,r1,r2) muszą być takie same

   Wzór na znak objętości tetrahedronu (a,b,c,d) = $sign((\vec{AB} \times \vec{AC}) \cdot \vec{AD})$. Boki i wierzchołki trójkąta nie powodują kolizji. Jeżeli odcinek się kończy lub zaczyna na trójkącie, zwracane jest true. W każdym innym wypadku zwraca false. Wykorzystywana w API w:
   - w procedurze `[countCollisionsWithMeshFromOutsideMeshToPoint] (@pointId int, @meshId int, @outsideX float, @outsideY float, @outsideZ float)`
5. `[calculateMeshVolume] (@meshString nvarchar(max))` - zwraca objętość (float) meshu. Jako wartość przyjmuje string opisujący manifold mesh: trójkąty oddzielone znakiem ;, punkty oddzielone znakiem , id i koordynaty punktu oddzielone znakiem ,. Najpierw string jest parsowany. Potem kolejność punktów wewnątrz trójkątów jest przestawiana tak, aby wszystkie wektory normalne trójkątów były skierowane na zewnątrz. Następnie oblicza sumę objętości $\frac{(\vec{AB} \times \vec{AC}) \cdot \vec{AD}}{6}$ tetrahedronów (a,b,c,d) o podstawie a,b,c w danym trójkącie i ostatnim wierzchołku d w dowolnym stałym punkcie w meshu. Zwracana jest wartość absolutna wyniku. Wykorzystywana w API w:
   - w procedurze `[getMeshVolume] (@meshId int)`
6. `[createPoint3D] (@x FLOAT, @y FLOAT, @z FLOAT)` - zwraca nowy `Point3D` o podanych koordynatach. Wykorzystywana w API w:
   - `int insertPoint(double x, double y, double z)`
   - w funkcji `[getPointBeyondMinimalMeshBox] (@meshId int)`
   - w procedurze `[countCollisionsWithMeshFromOutsideMeshToPoint] (@pointId int, @meshId int, @outsideX float, @outsideY float, @outsideZ float)`
7. `[isMeshIsManifold] (@triangles NVARCHAR(max))` - zwraca true, jeżeli mesh jest manifold - w przeciwnym wypadku false. Sprawdza to przez przyjęcie wartości string opisującej trójkąty, gdzie trójkąty oddzielone są znakiem ;, a id punktów wierzchołkowych w trójkącie znakiem −. String ten jest parsowany, a następnie sprawdzane jest, czy spełnione są 2 warunki:
   - wszystkie krawędzie muszą przystawać do dokładnie 2 trójkątów
   - wszystkie wierzchołki sąsiadujące z danym wierzchołkiem muszą być połączone w jeden nieprzerwany łańcuszek/cykl.

   Wykorzystywana w API w:
   - w funkcji `[checkIfMeshIsManifold] (@id int)`

Poniższe funkcje są zdefiniowane w T-SQL.

1. `[checkIfMeshIsManifold] (@id int)` - Funkcja konsolidująca informacje o trójkątach danego meshu i wysyłająca je do `[isMeshIsManifold] (@triangles NVARCHAR(max))`. Zwraca (bit) null, jeżeli mesh o danym id nie istnieje. False, jeżeli jest pusty. Wartość funkcji do której wysyła informacje w przeciwym wypadku. Wykorzystywana w API w:
   - w tabeli `Mesh` do obliczania wartości kolumny isManifold
2. `[getPointBeyondMinimalMeshBox] (@meshId int)` - zwraca punkt `Point3D` poza bounding boxem meshu. Jeżeli otoczylibyśmy mesh najmniejszym możliwym prostopadłościanem o bokach równoległych do osi XYZ, to wybierany jest wierzchołek o najmniejszych koordynatach i przesuwany o wektor [-1,-1,-1], następnie zwracany.

Zwraca null, jeżeli mesh nie istnieje. Wykorzystywana w API w:

- `bool isPointInsideMesh(int pointId, int meshId)`

3.4 Tabele

1. Point
   - id - int not null - primary key o samozwiększającej się wartości, indeksowany od 1
   - coordinates - Point3D not null - koordynaty punktu
2. Triangle
   - id - int not null - primary key o samozwiększającej się wartości, indeksowany od 1
   - point_id1 - int not null - wierzchołek trójkąta - foreign key do id w Point
   - point_id2 - int not null - wierzchołek trójkąta - foreign key do id w Point
   - point_id3 - int not null - wierzchołek trójkąta - foreign key do id w Point
   - id wierzchołków muszą być ustawione w kolejności rosnącej, a takie same 2 punkty nie mogą tworzyć dwóch trójkątów
3. Mesh
   - id - int not null - primary key o samozwiększającej się wartości, indeksowany od 1
   - isManifold - bit - kolumna obliczana z funkcji `[checkIfMeshIsManifold](id)`, określająca, czy mesh jest manifold
4. MeshTriangle - tabela asocjacyjna między Mesh i Triangle opisująca, które trójkąty należą do meshy
   - mesh_id - int not null - foreign key do id w Mesh - on delete cascade
   - triangle_id - nt not null - foreign key do id w Triangle - on delete cascade
   - (mesh_id, triangle_id) - primary key

3.5 Wyzwalacze

1. `[deleteTrianglesThatUsedDeletedPointTrigger]` - przed usunięciem punktu usuwa wszystkie trójkąty, które go używały
2. `[deleteMeshesThatHaveNoTrianglesTrigger]` - jeżeli po usunięciu trójkąta z meshu, mesh nie ma już żadnego trójkąta, to jest on także usuwany
3. `[resetIdentityOnPointTrigger]` - po usunięciu punktu ustawia identity w tabeli Point na maksymalne id (lub 0, jeżeli tabela jest pusta)
4. `[resetIdentityOnTriangleTrigger]` - po usunięciu trójkąta ustawia identity w tabeli Triangle na maksymalne id (lub 0, jeżeli tabela jest pusta)
5. `[resetIdentityOnMeshTrigger]` - po usunięciu meshu ustawia identity w tabeli Mesh na maksymalne id (lub 0, jeżeli tabela jest pusta)

3.6 Procedury

1. `[isPointOnMesh] (@pointId int, @meshId int)` - sprawdza, czy punkt jest na meshu. Zwraca null, jeżeli punkt lub mesh o podanych id nie isnieją. Zwraca 1, jeżeli punkt znajduje się wśród wierzchołków meshu. W każdym innym wypadku zwraca 1, jeżeli wartość funkcji `[checkIfPointInTriangle] (@p [Point3D], @a [Point3D], @b [Point3D], @c [Point3D])` jest prawdziwa dla podanego punku i jakiegokolwiek trójkąta w meshu. W przeciwnym wypadku zwraca 0. Wykorzystywana w API w:

- `bool isPointOnMesh(int pointId, int meshId)`

2. `[isPointOnTriangle] (@pointId int, @triangleId int)` - sprawdza, czy punkt jest na trójkącie. Zwraca null, jeżeli punkt lub trójkąt o podanych id nie isnieją. Zwraca 1, jeżeli punkt znajduje się wśród wierzchołków trójkąta. W każdym innym wypadku zwraca wartość funkcji `[checkIfPointInTriangle] (@p [Point3D], @a [Point3D], @b [Point3D], @c [Point3D])` dla podanego punku i trójkąta. Wykorzystywana w API w:
   - `bool isPointOnTriangle(int pointId, int triangleId)`
3. `[getMeshArea] (@meshId int)` - zwraca pole powierzchni meshu. Pole każdego trójkąta liczone jest tylko raz. Zwraca null, jeżeli mesh nie istnieje. Zwraca 0, jeżeli mesh nie ma żadnych trójkątów. Zwraca sumę z wywołań funkcji `[getTriangleArea] (@a [Point3D], @b [Point3D], @c [Point3D])` dla każdego trójkąta w meshu. Wykorzystywana w API w:
   - `double getMeshArea(int id)`
4. `[getMeshVolume] (@meshId int)` - zwraca objętość meshu. Zwraca null, jeżeli mesh nie jest manifold. Konsoliduje dane o meshu do stringu i wywołuje dzięki niemu funkcję `[calculateMeshVolume] (@meshString nvarchar(max))`, której wartość zwraca. Wykorzystywana w API w:
   - `double getMeshVolume(int id)`
5. `[countCollisionsWithMeshFromOutsideMeshToPoint] (@pointId int, @meshId int, @outsideX float, @outsideY float, @outsideZ float)` - liczy, ile trójkątów danego meshu przeciął odcinek od punktu `pointId` do punktu o koordynatach `outsideX, outsideY, outsideZ`. Zwracana jest suma wywołań `[doesRayIntersect] (@a [Point3D], @b [Point3D], @c [Point3D], @r1 [Point3D], @r2 [Point3D])` o wartościach true dla każdego trójkąta meshu, punktu `pointId` i punktu o koordynatach `outsideX, outsideY, outsideZ`. Wykorzystywana w API w:
   - `bool checkIfPointIsInsideMesh(SqlConnection openConnection, int pointId, int meshId, double x, double y, double z)` - jeżeli ilość kolizji była nieparzysta, to punkt jest wewnątrz meshu (liczone jest to kilka razy dla różnych punktów, ponieważ na krawędziach i wierzchołkach kolizje nie są wykrywane)

4 Testy API

Aby przetestować API, przeprowadzono głównie testy integracyjne. Przed wszystkimi testami definiowany jest connection string do bazy danych oraz obiekt klasy TriangleMeshAPI. Przed kazdym testem otwierana jest nowa transakcja, a baza danych jest czyszczona. Transakcja zamykana jest po wykonaniu testu.

Testy integracyjne dla funkcji:

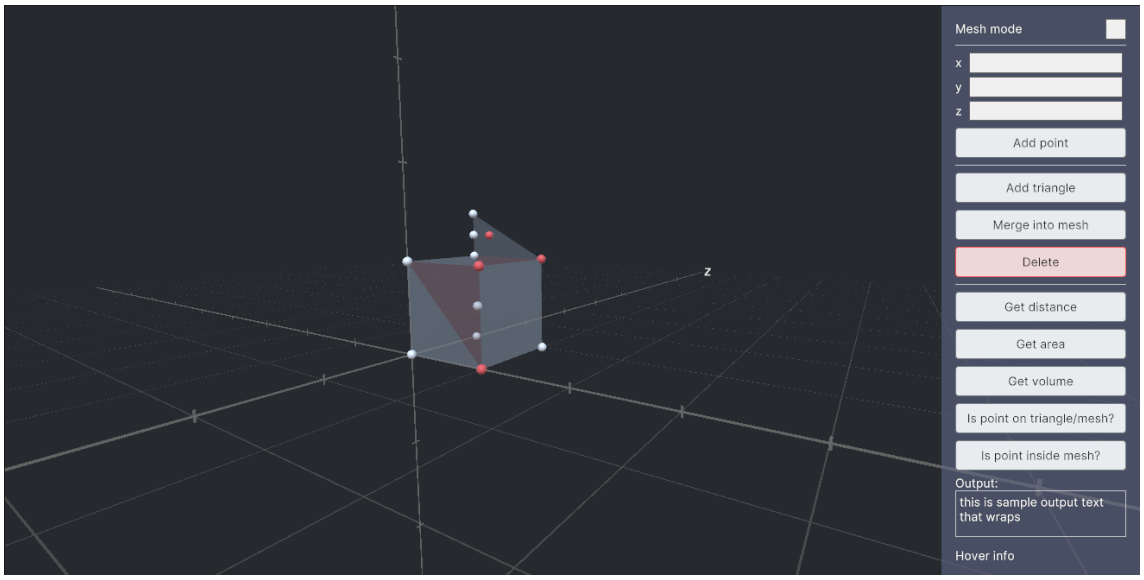
1. `fetchPoints, fetchTriangles, fetchMeshes` - testują zwracanie poprawnych wartości dla wielu obiektów, dla zera obiektów oraz wyrzucanie błędów, gdy do funkcji zostaną przekazane niekompletne lub błędne (powtarzające się id) listy z referencjami do obiektów
2. `insertPoint, insertTriangle, insertMesh` - testują, czy zwracane są poprawne id przy wstawianiu wielu obiektów; czy da się wstawiać liczby ułamkowe; czy wyrzucane są błędy przy wstawianiu powtarzających się trójkątów, przy wstawianiu do meshu(/trójkąta) trójkątów(/punktów) z nieistniejącymi/powtarzającymi się id
3. `mergeMeshes` - testują, czy łączenie meshy w jeden tworzy nowy mesh, usuwa użyte i zwraca aktualną listę meshy; czy działa łączenie meshy wykorzystujących te same trójkąty; czy łączenie 0 lub 1 meshu nic nie zmienia; czy wyrzucany jest błąd przy podaniu niekompletnej listy referencji do trójkątów
4. `deletePoints, deleteTriangles, deleteMeshes` - testują poprawne działanie usuwania wielu obiektów; czy zwracane są poprawne listy aktualnych danych; czy poprawnie resetuje

się indeksacja przy usuwaniu kilku i wszystkich obiektów; czy nie jest wyrzucany błąd przy usuwaniu obiektów o id z pustej listy; czy usuwanie jednych obiektów poprawnie wpływa na nieusuwanie innych; czy wyrzucany jest błąd przy podaniu niekompletnej listy referencji do obiektów

5. `getDistanceBetweenPoints` - testują, czy zwracana jest poprawna wartość dla różnych i tych samych punktów; czy wyrzucany jest błąd przy podaniu id nieistniejących punktów
6. `getTriangleArea, getMeshArea` - testują, czy zwracana jest poprawna wartość; czy zwracane jest 0 dla trójkątów, które są liniami; czy zwracane jest 0 dla pustych meshy; czy wyrzucany jest błąd przy podaniu id nieistniejących obiektów
7. `isMeshManifold` - testują, czy poprawnie identyfikowana jest manifold mesh; czy poprawnie wykrywane są 3 przypadki, kiedy nie jest; czy zwracany jest fałsz dla pustych meshy; czy wyrzucany jest błąd przy podaniu id nieistniejącego obiektu
8. `getMeshVolume` - testują, czy zwracana jest poprawna wartość; czy zwracane jest 0 dla płaskich meshy; czy wyrzucany jest błąd przy podaniu id nieistniejącego/non-manifold meshu
9. `isPointInsideMesh` - testują oba przypadki - dla punktu poza i wewnątrz meshu oraz czy wyrzucany jest błąd przy podaniu id nieistniejącego/non-manifold meshu lub id nieistniejącego punktu
10. `isPointOnTriangle, isPointOnMesh` - testują oba przypadki - dla punktu bardzo blisko i daleko od powierzchni trójkąta/meshu oraz czy zwracana jest prawda, gdy punkt jest jednym z wierzchołków tych struktur oraz gdy jest na jednej z krawędzi tych struktur; czy wyrzucany jest błąd przy podaniu id nieistniejącego obiektu.

Testy jednostkowe sprawdzają jedynie poprawne tworzenie struktur Point, Triangle i Mesh oraz poprawność działania ich metod ToString().

5 Przykładowe wykorzystanie API - aplikacja



Rysunek 1: Wygląd aplikacji implementującej API.

TriangleMeshSandbox to przykładowa aplikacja napisana w silniku Unity, wykorzystująca metody i typy danych udostępniane przez TriangleMeshAPI. Pozwala ona na poruszanie się w przestrzeni 3D z zaznaczonymi osiami XYZ w centrum układu oraz wyświetlanie i manipulację danymi w bazie danych przez interfejs łączący się z API przez DLL.

Użytkownik może:

- poruszać się w przestrzeni 3D trzymając prawy przycisk myszy, poruszając myszką i wciskając klawisze AWSD
- przyspieszyć poruszanie się przez przytrzymanie klawiszu Shift
- precyzyjnie ustawiać kamerę poruszając myszką z wciśniętym środkowym przyciskiem myszy
- resetować kamerę klawiszem R
- klikać na punkty/trójkąty lewym przyciskiem myszy, aby je zaznaczyć (lub trzymając lewy Alt, aby je odznaczyć)
- kliknąć lewym przyciskiem myszy na pustą przestrzeń trzymając lewy alt, aby odznaczyć wszystkie obiekty
- najeżdżać na punkty/trójkąty, aby wyświetlać ich id w bazie danych w prawym dolnym rogu ekranu. Najechane obiekty przyjmują kolor niebieski, a ich sąsiedzi jasno-niebieski.
- zaznaczyć w UI checkbox Mesh Mode, aby wejść w tryb wyświetlania meshy, podczas którego może strzałkami w prawo i w lewo cyklować pomiędzy wyświetlanymi meshami. Wyświetlany mesh jest koloru pomarańczowego i także może być zaznaczony lewym przyciskiem myszy, tak jak punkty. Wchodzenie i wychodzenie z Mesh Mode odznacza wszystkie obiekty.
- dodać punkt przez wpisanie jego koordynatów i zatwierdzenie przyciskiem. Przed zatwierdzeniem podgląd punktu wyświetla się na zielono w przestrzeni 3D.
- dodać trójkąt na bazie 3 zaznaczonych punktów
- dodać mesh na bazie wielu zaznaczonych trójkątów
- połączyć zaznaczone meshe w jeden
- usunąć wszystkie zaznaczone obiekty
- sprawdzić dystans pomiędzy dwoma punktami, powierzchnię zaznaczonego trójkąta, powierzchnię zaznaczonego meshu, objętość zaznaczonego meshu
- sprawdzić, czy zaznaczony punkt jest w środku zaznaczonego meshu oraz czy jest na powierzchni zaznaczonego trójkąta/meshu
- przeczytać wyniki różnych operacji w polu „Output" w prawym dolnym rogu ekranu.
- zamknąć aplikację klawiszem Escape

6 Podsumowanie

Triangle Mesh Sandbox to aplikacja, której cała logika (poza obsługą akcji użytkownika) odbywa się wewnątrz bazy danych. Wykonanie bardziej skomplikowanych operacji jest możliwe i stosunkowo wygodne dzięki wykorzystaniu technologii CLR - w tym wypadku w szczególności UDT, UDF i UDA. Dzięki odpowiedniemu oznaczeniu tych obiektów CLR, SQL Server jest w stanie odpowiednio indeksować wyniki wywołania funkcji, agregatów i zapytań, i przez to znacznie przyspieszyć wykonywanie obliczeń.

7 Linki (kody źródłowe)

Github z kodami źródłowymi można znaleźć pod linkiem `https://github.com/aSliwska/triangle-mesh-sandbox`. Znajduje się tam także instrukcja własnej kompilacji lub używania DLL API, uruchomienia aplikacji i testów oraz postawienia bazy danych ze skryptu T-SQL i plików DLL.

Poszczególne elementy projektu można znaleźć w tych folderach:

- API - api/solution/API
- aplikacja - aplikacja/TriangleMeshSandbox/TriangleMeshSandbox
- testy - api/solution/APITest
- skrypty CLR - api/solution/SPS_scripts
- skrypty T-SQL - api/tsql

8 Literatura

- Odpowiedzi na różne pytania użytkowników na stronach
  - `https://math.stackexchange.com`
  - `https://stackoverflow.com`
  - `https://dba.stackexchange.com`
  - `https://gamedev.stackexchange.com`
  - `https://computergraphics.stackexchange.com/`
  - `https://softwareengineering.stackexchange.com/`
  - `https://physics.stackexchange.com/`
  - `https://forum.unity.com/`
  - `https://discussions.unity.com/`
- Dokumentacja Unity - `https://docs.unity3d.com/`
- Unity Camera Controls script - `https://forum.unity.com/threads/how-to-make-camera-move-in-a-way-si 524645/`
- Unity Grid Shader script - `https://forum.unity.com/threads/wireframe-grid-shader. 60071/`