Assignment 5

Due Date and Time: Please refer to the course web page.

Topics

- Scan-conversion techniques for lines, circles, triangles, and convex polygons
  - Identifying clockwise or counter-clockwise orientation of triangles
  - Classifying an arbitrary polygon as convex or concave
  - Attribute interpolation for triangles using linear interpolation
- Bounding area construction
  - Bounding circle (BC)
  - Axis-aligned bounding box (AABB)
- 2D Picking
  - Point in BC test
  - Point in AABB test
  - Point in convex polygon test
- Hierarchical object intersections-
  - BC vs. BC
  - AABB vs. AABB
  - BC vs. AABB
  - AABB vs. convex polygon
  - BC vs. convex polygon
  - Convex polygon vs. convex polygon

Programming Section: Statement

Note that A5 framework is implemented in left-handed device frame while discussions in class were in world frame. Therefore, certain mathematical constructs such as outward normal to an edge are computed differently in this application compared to class notes.

Implement the functions specified below. See corresponding function header for additional implementation details. As functions are correctly implemented, your screen must eventually look similar to the enclosed sample executable.

- To render circular pillars, complete function definition: void CircularPillar::Draw(const Window&) const;
- Your screen should now be displaying circular pillars described in screen file.
- To render polygonal pillars, complete the following polygon scan-conversion function definitions:
  - To scan-convert filled counter-clockwise oriented triangles: void Triangle::Draw(const Window&) const;
  - To scan-convert filled counter-clockwise oriented convex polygons: void PolygonalPillar::Draw(const Window&) const;
- Next, consider bounding areas enclosing solid pillars.
  - To compute BC of each solid pillar, complete function definition: void BC::Compute(const Point2D*, int);
  - To render boundaries of bounding circles, complete function definition: void BC::Draw(const Window&) const;

    - To compute AABB of each solid pillar, complete function definition: void AABB::Compute(const Point2D*, int);
    - To render AABB boundaries, complete following function definitions:
        - void Line2D::Draw(const Window&) const;
        - void AABB::Draw(const Window&) const;
- At this stage, your screen will be similar to the one displayed by the sample executable. You should also be able to create new circular and rectangular pillar using Mode menu's Create selection. The next step is to correctly identify triangles as clockwise or counter-clockwise oriented and to classify polygons as convex or concave.
    - To compute point-normal equation of polygonal edges, complete function definition: void Edge2D::ComputePointNormalEquation();
    - To determine clockwise or counter-clockwise triangle orientation, complete function definition: bool Triangle::IsCounterClockwise() const;
    - To determine polygon convexity or concavity, complete function definition: bool PolygonalPillar::IsConvex() const;
- 2D picking is incorporated into the application by implementing the following point-in-object tests. Note that when a pillar is picked, the mouse icon changes from IDC_CROSS to IDC_SIZEALL.
    - Point in circle test: bool CircularPillar::TestPillarVsPoint(const Point2D&) const;
    - Point in convex polygon test: bool PolygonalPillar::TestPillarVsPoint(const Point2D&) const;
- When the mouse is dragged, the picked pillar is displaced by relative changes in mouse cursor location from one frame to next. To update the pillar and its bounding areas, complete the following functions:
    - void CircularPillar::Move(const Vector2D&);
    - void PolygonalPillar::Move(const Vector2D&);
    - void BC::Move(const Vector2D&);
    - void AABB::Move(const Vector2D&);
- To ensure correct and robust simulations, a picked pillar cannot be repositioned to a new location if at that position:
    - The pillar penetrates another pillar.
    - The pillar is completely contained within another pillar.
    - The pillar is completely containing another pillar.
- To detect intersection and containment, the definitions of the following inter-object intersection function must be completed:
    - bool BC::TestBCVsPoint(const Point2D&) const;
    - bool BC::TestBCVsBC(const BC&) const;
    - bool TestBCVsAABB(const BC&, const AABB&);
    - bool AABB::TestAABBVsPoint(const Point2D&) const;
    - bool AABB::TestAABBVsAABB(const AABB&) const;
    - bool CircularPillar::TestPillar(const Pillar*) const;
    - bool PolygonalPillar::TestPillarVsBC(const BC&) const;
    - bool PolygonalPillar::TestPillarVsPP(const PolygonalPillar&) const;
    - bool PolygonalPillar::TestPillar(const Pillar*) const;

Programming Section: Statement

Provide a *manual* page with your submission. This must include your derivations/discussions on the mathematical aspects of the implemented code. For this assignment, the specific details to be included are:

- AABB construction.
- BC construction.
- Point vs. AABB intersection test.
- Point vs. BC intersection test.
- Point vs. convex polygon intersection test.
- BC vs. BC intersection test.
- BC vs. AABB intersection test.
- BC vs. convex polygon intersection test.
- Convex polygon vs. convex polygon intersection test using Separation Axis Theorem.

In your own words, describe what has been implemented, how it was implemented, any modifications made to the assignment specifications, cool features implemented, etc. Finally, specify any portion of the assignment that is incomplete or not attempted.

Deliverables:

- Zipped folder containing the relevant documents (derivation, README file, declaration page) and containing entire framework (object code and executables must be deleted). We will unzip your folder, double-click on "A5.sln" to open, compile, link, and execute your project.

Objectives: CS200 A5 (Worth 100 points)

Name: _____

Login Name: _____

Student ID: _____

___1: Correct information was handed in. If not, penalty is **–20** points.
___2: Student program compiles, links and executes. If not, penalty is **–100** points.
___3: Filled circular pillars, filled polygonal pillars, BC boundaries, and AABB boundaries are rendered correctly. If not, penalty is **–20** points.
___4: Create selection of Mode menu discards clockwise triangles. Worth **5** points.
___5: Create selection of Mode menu discards concave polygons. Worth **5** points.
___6: BCs and AABBs are constructed with the tightest fit possible out of all the methods discussed in class. Worth **10** points.
___7: Object picker is implemented correctly for circular pillars: Worth **5** points.
___8: Object picker is implemented correctly for polygonal pillars using hierarchical tests starting with enclosing BC, enclosing AABB, and ending with inside-outside tests with polygon edges. Worth **15** points.
___9: Object picker allows users to reposition picked pillar by dragging the mouse. For robust simulations, a pillar cannot be allowed to straddle other pillars, contain other pillars, or be contained by other pillars. Worth **50** points.
___a) Intersection between circular pillars (or, BCs). Worth **5** points.
___b) Intersection between circular pillar (or, BC) and rectangular pillar (or, AABB). Worth **10** points.
___c) Intersection between rectangular pillars (or, AABBs). Worth **5** points.
___d) Intersection between circular pillar (or, BC) and polygonal object. Worth **15** points.
___e) Intersection between rectangular pillar (or, AABB) and polygonal pillar and intersection between polygonal pillars. Worth **15** points.
___10: A good manual page was submitted. Worth **10** points.

DECLARATION:
I have read the statements regarding cheating in both the CS200 course handout and DigiPen student handbook. I affirm with my signature that this is my own solution to A5 and the submitted source code and manual are of my creation and represent my own work.

Signature: