Physics 11

CITM

Box2d - Shapes

Box2D Shapes

Box2D supports different shapes:

- *Circles*
- Convex Polygons
- Edges
- Chains

Box2D Shapes: Polygons

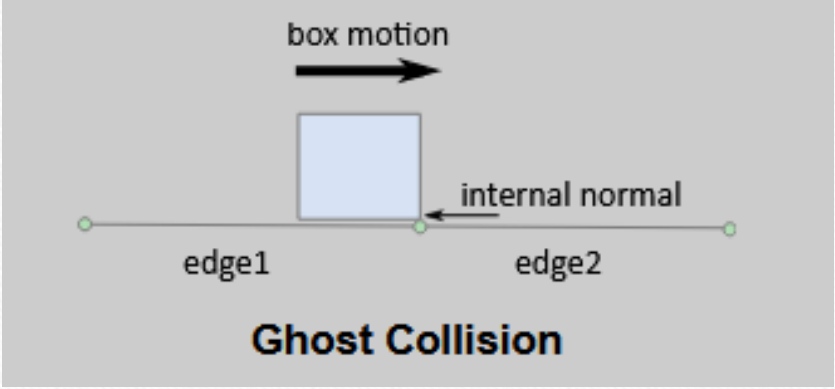
- Only **convex** polygons!
- They are never hollow.
- The vertices are declared in counterclockwise order.
- The maximum amount of vertices is 8! (*b2_maxPolygonVertices*).
- We have convenient functions to create rectangles:

```
void SetAsBox(float32 hx, float32 hy);

b2PolygonShape box;
box.SetAsBox(…);
```

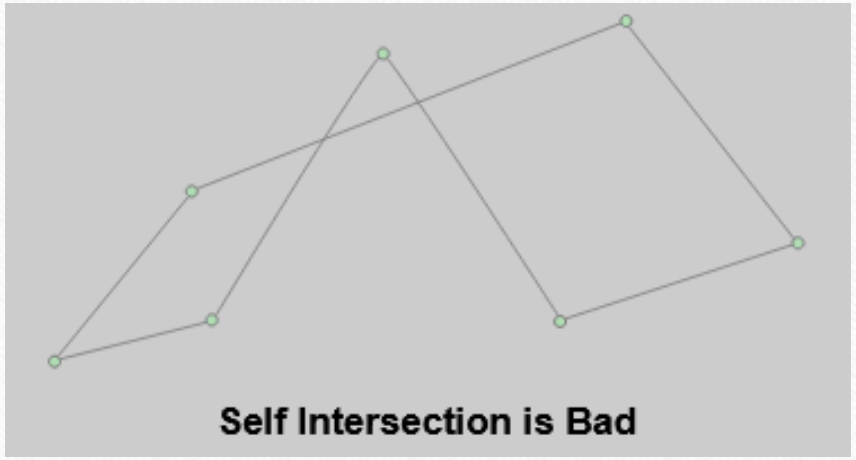
Box2D Shapes: Edges

- Edges are just a **line segments**
- Used to give contour to levels
- They have no volume
- They cannot collide with themselves
- Beware of the ghost collisions!



Box2D Shapes: Chains

- They are used to connect edges
- You usually do not create Edges alone
- Chains prevent ghost collisions
- Self-colliding chains is not supported

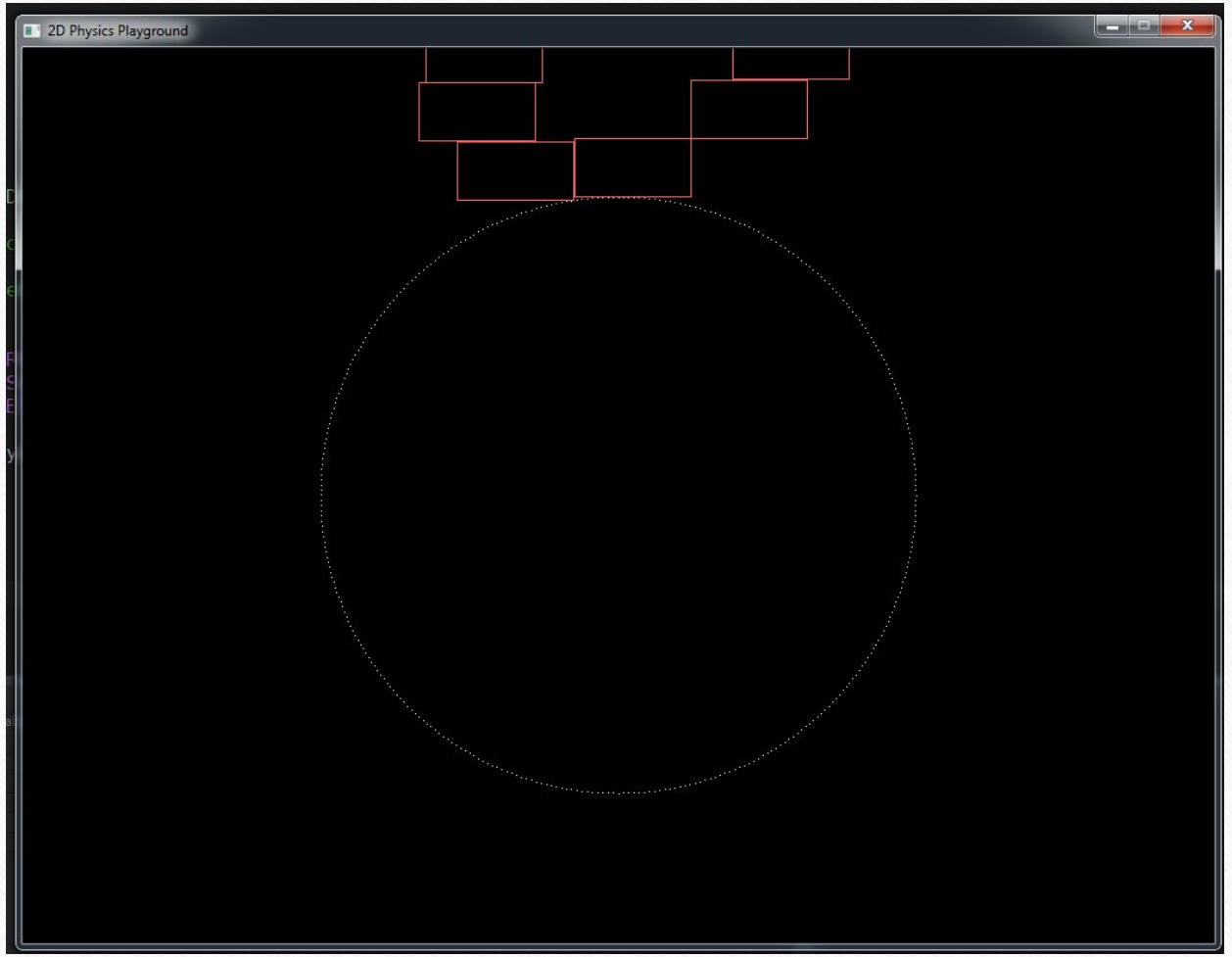


TODO 1

When pressing 2, create a box on the mouse position

- Copy & paste the code to create a circle
- Just change the shape to a rectangle

Weird!



TODO 2

To have the box behave normally, set fixture's density to 1.0

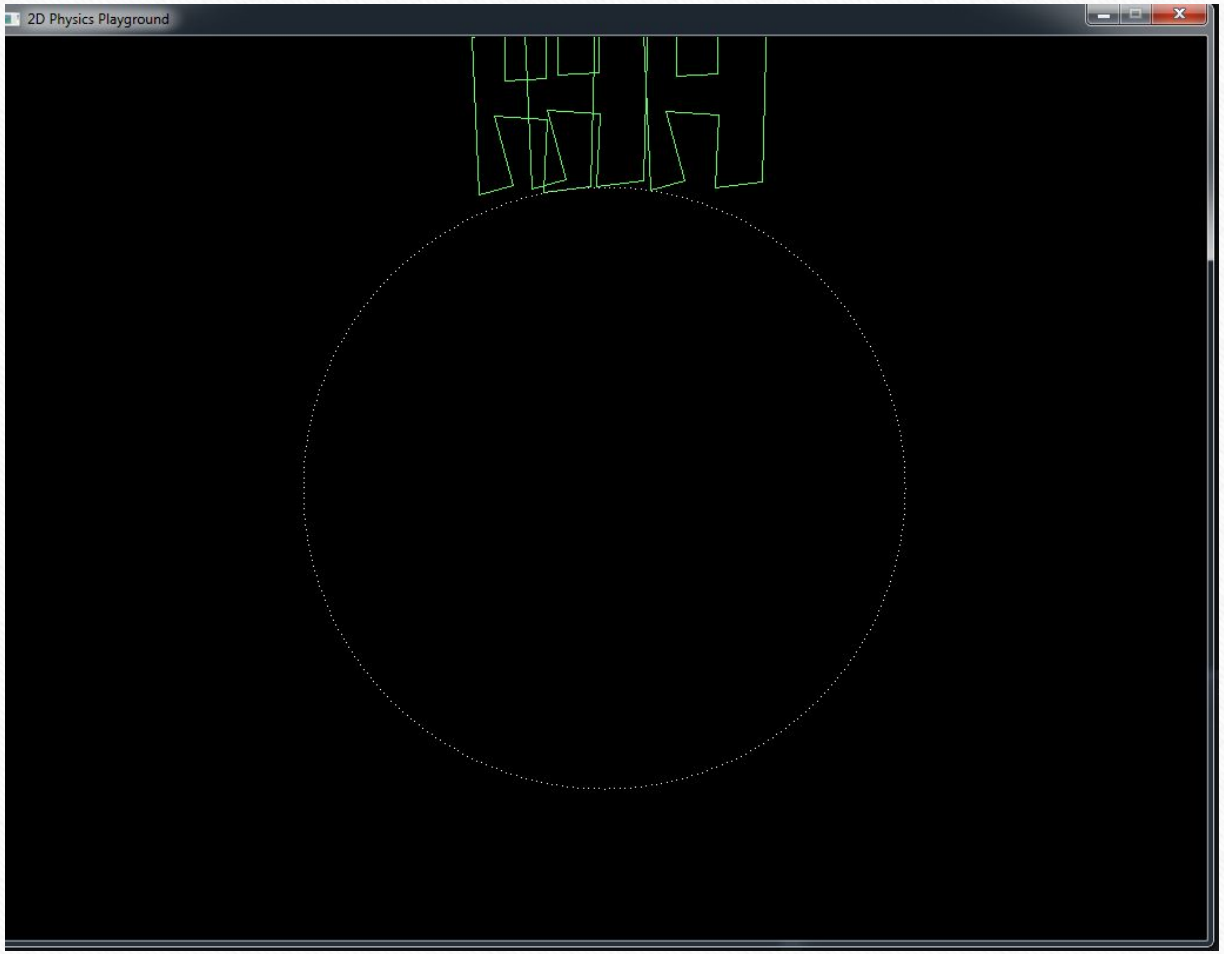
- See what happens then

TODO 3

If user presses 3, create a chain with those vertices

- You need to fill an array of b2Vec2
- Remember to transform all points
- Then call “shape.createLoop()” method

Chains do not have volume!



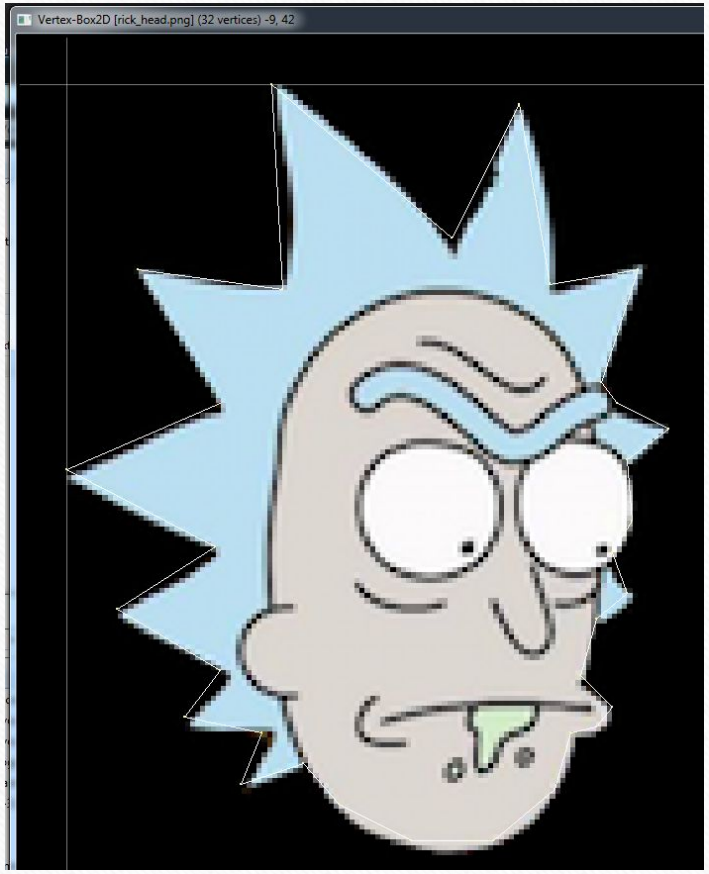
Our own shapes

- Download my awesome shape creator:

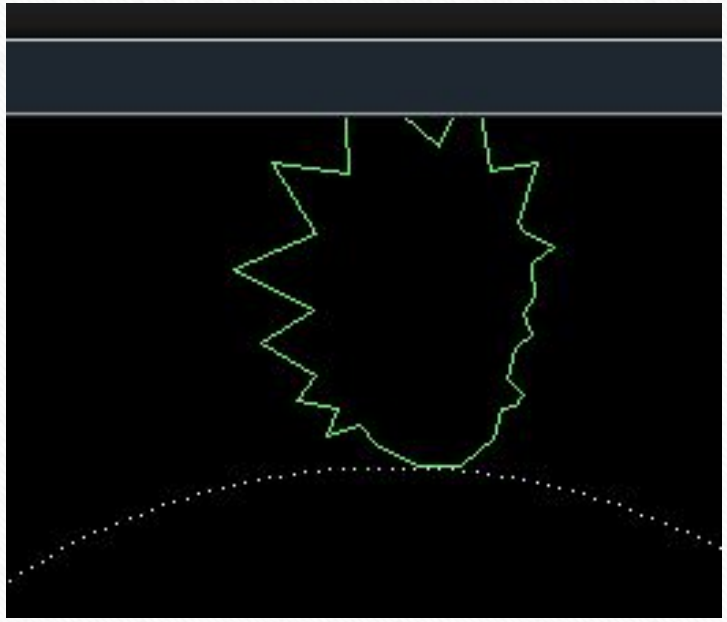
  https://d0n3val.github.io/Vertex-Box2D/

- Drag in Rick_head.png from the Game/pinball directory
- Draw the approximate edges of the picture
- Do not close it, leave the last segment!
- Press “ESC” and it will drop the info to paste buffer and close itself

Creating your own shape



Now try using those vertices instead of the previous ones, you should get something like this:



TODO 4

Move body creation to 3 functions to create circles, rectangles and chains

- Just moving code
- Functionality stays the same, just cleaner:

```
if(App->input->GetKey(SDL_SCANCODE_1) == KEY_DOWN) {
    CreateCircle(App->input->GetMouseX(), App->input->GetMouseY(), 50);
}
```

TODO 5

Move all creation of bodies on 1,2,3 key press here in the scene

- Again, just moving code
- Functionality stays the same, but on **ModuleSceneIntro.cpp**
- You can remove that debug code from **ModulePhysics.cpp**

Now we want to draw textured objects!

- We need to return **something** when creating a physical body
- That **something**, we should be able to request its position
- We can create a small class that stores the body
- …and has a method to request its position.

TODO 6

Create a small class that keeps a pointer to the **b2Body** and has a method to request the position then write the implementation in the .cpp

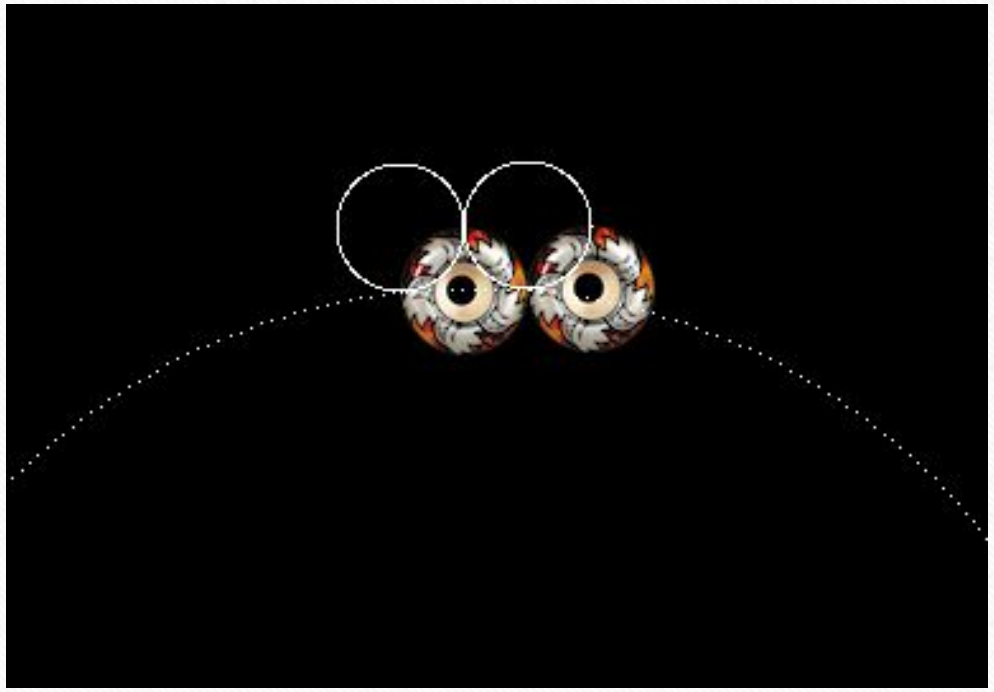
- Declaration can happen in **ModulePhysics.h**
- Write its definition in **ModulePhysics.cpp**
- Check b2Body.GetPosition()
- You have to convert the data you receive from meters to pixels!
- Have your circle creation method to return a pointer to this class

TODO 7

Draw all the circles using **circle** texture

- Declare a list/array to store all bodies
- Every time we create a circle pressing 1, add it to your list/array
- By the end of Update() method, iterate all circles and draw them
- First request their position, then use **circle** texture already loaded

You should get something like this



- The position we get is the center of the circle, but we draw from top-left!

Homework

- Solve the problem of drawing from the center
- Add a "GetRotation()" to our new mini class
- Draw the circle following the body rotation (use RADTODEG)
  - Render->Blit has a new argument that accepts rotation in **degrees**
- Now draw boxes and chains in the same way we draw circles
- You can check solution.exe to test the expected result
- BONUS: Allow for the creation of static and kinematic objects
- BONUS: clean up the memory