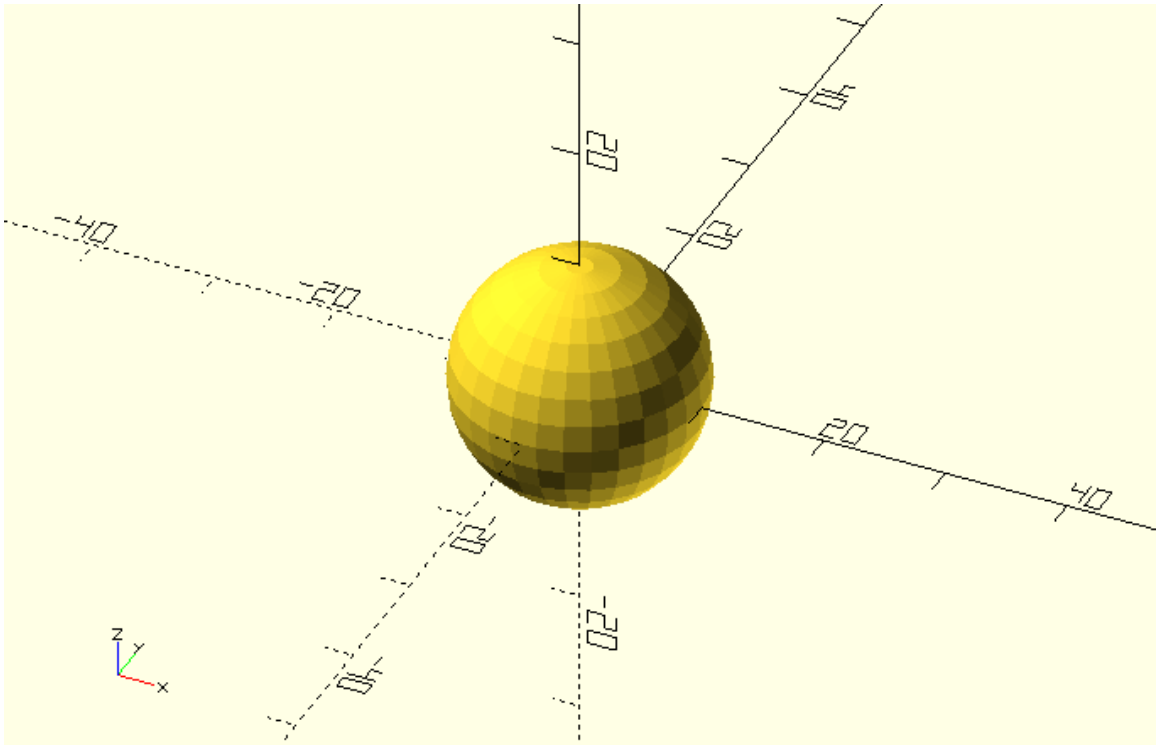
Chapter 3

The sphere primitive and resizing objects

You showed the car to your friends and they were quite impressed with your new skills. One of them even challenged you to come up with different futuristic wheel designs. It's time to put your creativity to work and learn more OpenSCAD features!

So far you have been using the cube and cylinder primitives. Another 3D primitive that is available in OpenSCAD is the sphere. You can create a sphere using the following command.

sphere(r=10);



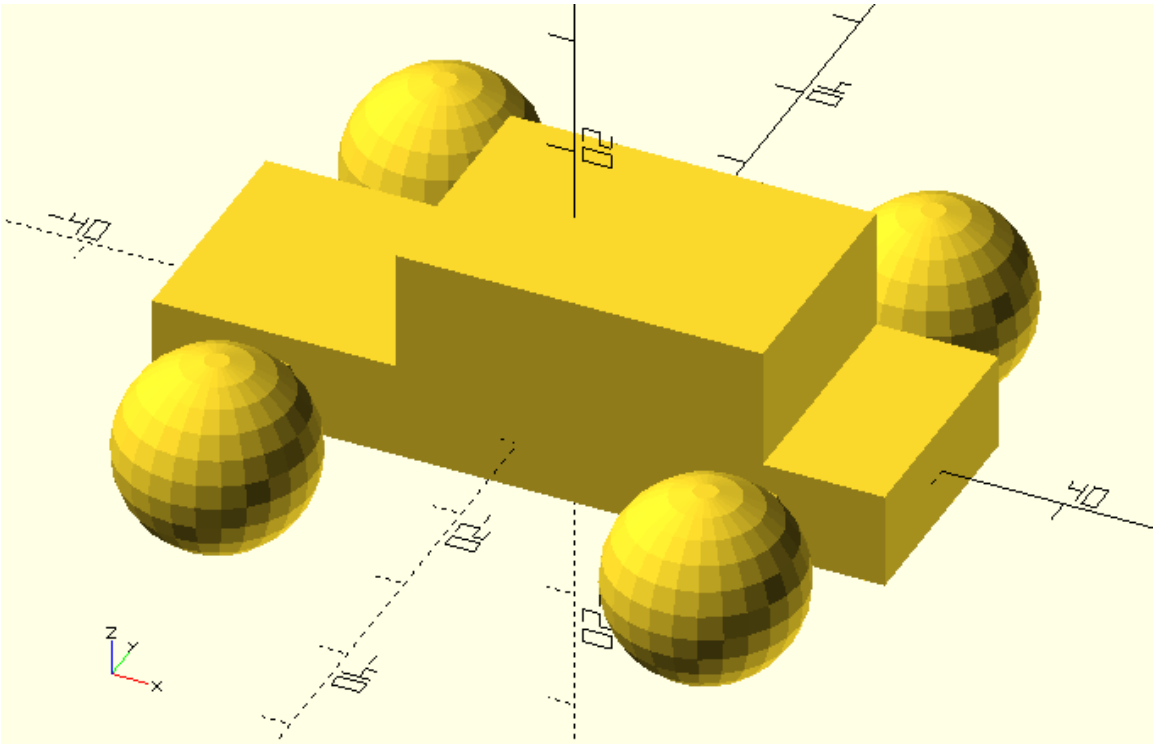
You should notice that the sphere is created centered on the origin. The input parameter r corresponds to the radius of the sphere.

One idea that came to your head was to replace the cylindrical wheels with circular ones.

Try making the wheels of your car spherical. To do so replace the appropriate cylinder commands with sphere commands. Is there still a need to rotate the wheels around the X axis? Is the wheel_width variable still required? Is there any visible change to your model when you modify the value of wheels_turn variable?

```
wheel_radius = 8;

base_height = 8;

top_height = 10;
```

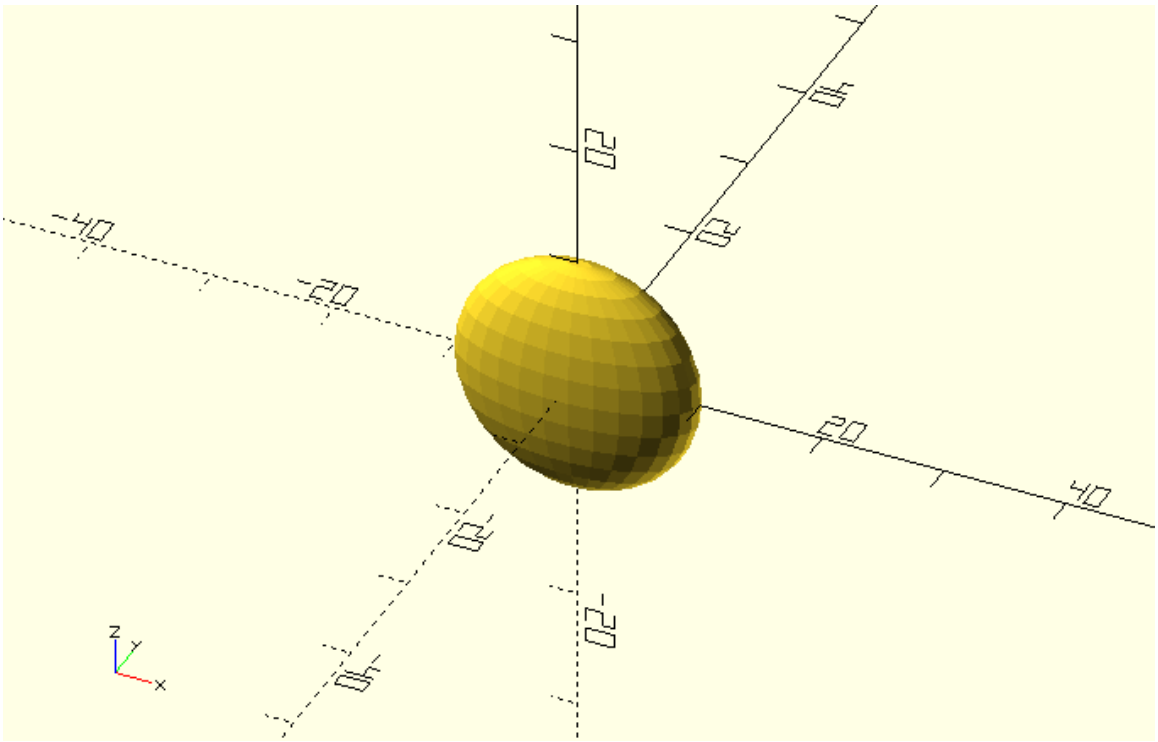
```
track = 40;
body_roll = 0;
wheels_turn = 20;
rotate([body_roll,0,0]){
// Car body base
cube([60,20,base_height],center=true);
// Car body top
translate([5,0,base_height/2+top_height/2])cube([30,20,top_height],center=true);
}
// Front left wheel
translate([-20,-track/2,0])rotate([0,0,wheels_turn])sphere(r=wheel_radius);
// Front right wheel
translate([-20,track/2,0])rotate([0,0,wheels_turn])sphere(r=wheel_radius);
// Rear left wheel
translate([20,-track/2,0])rotate([0,0,0])sphere(r=wheel_radius);
// Rear right wheel
translate([20,track/2,0])rotate([0,0,0])sphere(r=wheel_radius);
// Front axle
translate([-20,0,0])rotate([90,0,0])cylinder(h=track,r=2,center=true);
// Rear axle
translate([20,0,0])rotate([90,0,0])cylinder(h=track,r=2,center=true);
```



The idea to use a sphere to create the wheels was nice. You can now squish the spheres to give them a more wheel like shape. One way to do so is using the scale command.

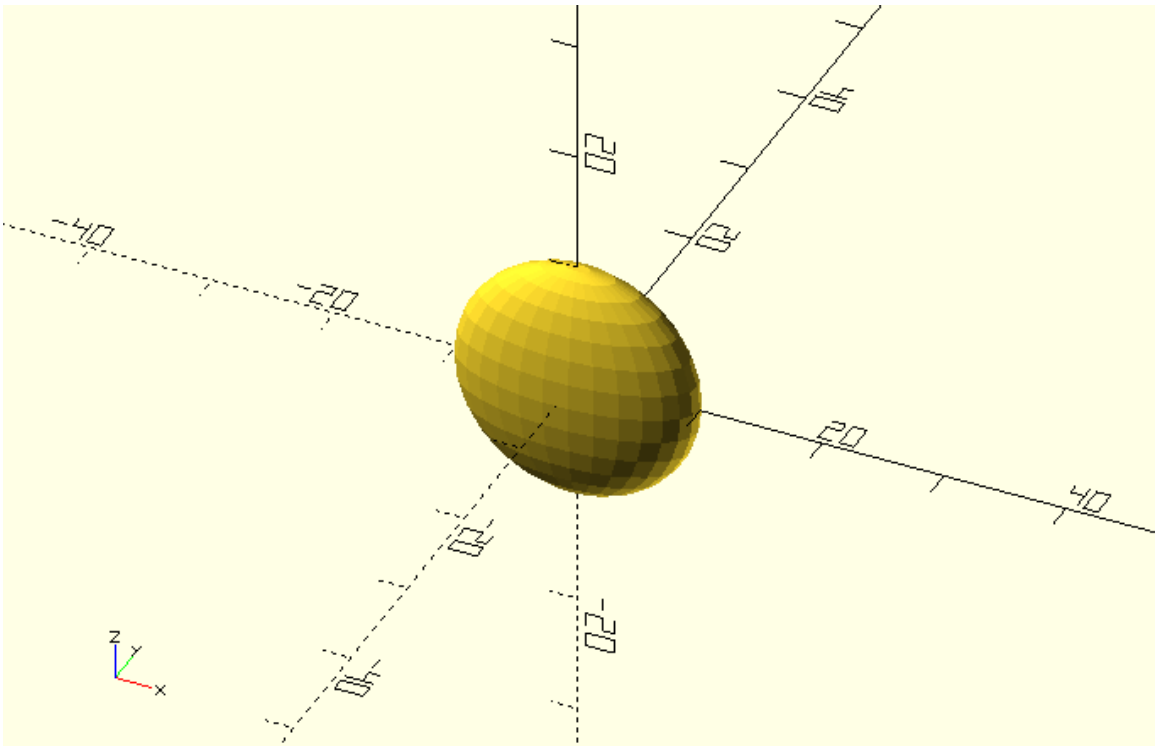
Try creating a sphere with a radius of 10 units on a blank model. Use the scale command to scale the sphere by a factor of 0.4 only along the Y axis.

```
scale([1,0.4,1])sphere(r=10);
```



Another way to scale objects is by using the resize transformation. The difference between scale and resize is that when using the scale command, you have to specify the desired scaling factor along each axis but when using the resize command you have to specify the desired resulting dimensions of the object along each axis. In the previous example you started with a sphere that has a radius of 10 units (total dimension of 20 units along each axis) and scaled it by a factor of 0.4 along the Y axis. Thus, the resulting dimension of the scaled sphere along the Y axis is 8 units. The dimensions along the X and Z axis remain the same (20 units) since the scaling factors along these axes are equal to 1. You could achieve the same result using the following resize command.

```
resize([20,8,20])sphere(r=10);
```

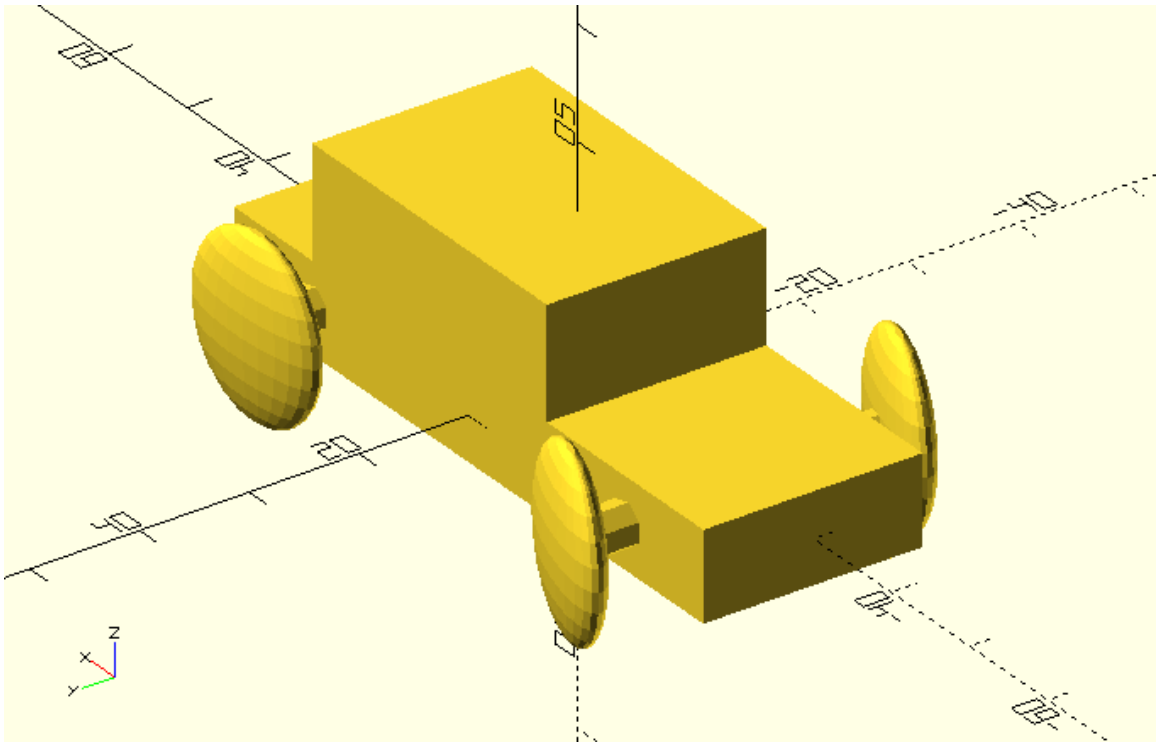


When you are scaling/resizing an object and you are concerned about its resulting dimensions it is more convenient to use the resize command. In contrast when you are concerned more about the ratio of the resulting dimensions compared to the starting dimensions it is more convenient to use the scale command.

Try squishing the spherical wheels of your car along the Y axis. Use the resize command and the wheel_width variable to have control over the resulting width of the wheels. Resize the wheels only along the Y axis.

```
wheel_radius = 8;

base_height = 8;

top_height = 10;

track = 30;

wheel_width = 4;

body_roll = 0;

wheels_turn = -20;

rotate([body_roll,0,0]){

// Car body base
```

```
cube([60,20,base_height],center=true);

// Car body top

translate([5,0,base_height/2+top_height/2])cube([30,20,top_height],center=true);

}

// Front left wheel

translate([-20,-
track/2,0])rotate([0,0,wheels_turn])resize([2*wheel_radius,wheel_width,2*wheel_radius])spher
e(r=wheel_radius);

// Front right wheel

translate([-
20,track/2,0])rotate([0,0,wheels_turn])resize([2*wheel_radius,wheel_width,2*wheel_radius])sp
here(r=wheel_radius);

// Rear left wheel

translate([20,-
track/2,0])rotate([0,0,0])resize([2*wheel_radius,wheel_width,2*wheel_radius])sphere(r=wheel_
radius);

// Rear right wheel

translate([20,track/2,0])rotate([0,0,0])resize([2*wheel_radius,wheel_width,2*wheel_radius])sp
here(r=wheel_radius);

// Front axle

translate([-20,0,0])rotate([90,0,0])cylinder(h=track,r=2,center=true);

// Rear axle

translate([20,0,0])rotate([90,0,0])cylinder(h=track,r=2,center=true);
```



The new wheel design looks cool. You can now create a body that better suits this new style.

Try using the sphere and resize/scale commands in place of the cube commands to create a body that matches the style of the wheels.

```
wheel_radius = 8;

base_height = 8;

top_height = 10;

track = 28;

wheel_width = 4;

body_roll = 0;

wheels_turn = -20;

rotate([body_roll,0,0]){

// Car body base

resize([90,20,12])sphere(r=10);

// Car body top

translate([10,0,5])resize([50,15,15])sphere(r=10);

}
```

// Front left wheel

translate([-20,-track/2,0])rotate([0,0,wheels_turn])resize([2*wheel_radius,wheel_width,2*wheel_radius])sphere(r=wheel_radius);

// Front right wheel

translate([-20,track/2,0])rotate([0,0,wheels_turn])resize([2*wheel_radius,wheel_width,2*wheel_radius])sphere(r=wheel_radius);

// Rear left wheel

translate([20,-track/2,0])rotate([0,0,0])resize([2*wheel_radius,wheel_width,2*wheel_radius])sphere(r=wheel_radius);

// Rear right wheel

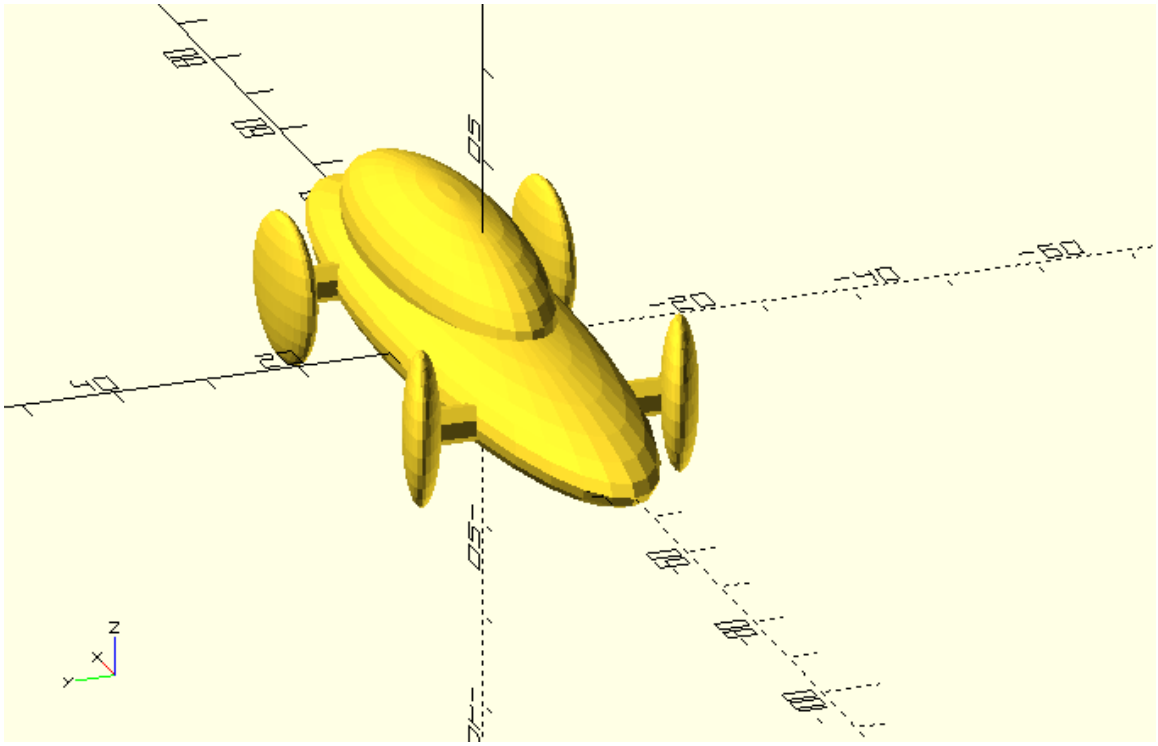
translate([20,track/2,0])rotate([0,0,0])resize([2*wheel_radius,wheel_width,2*wheel_radius])sphere(r=wheel_radius);

// Front axle

translate([-20,0,0])rotate([90,0,0])cylinder(h=track,r=2,center=true);

// Rear axle

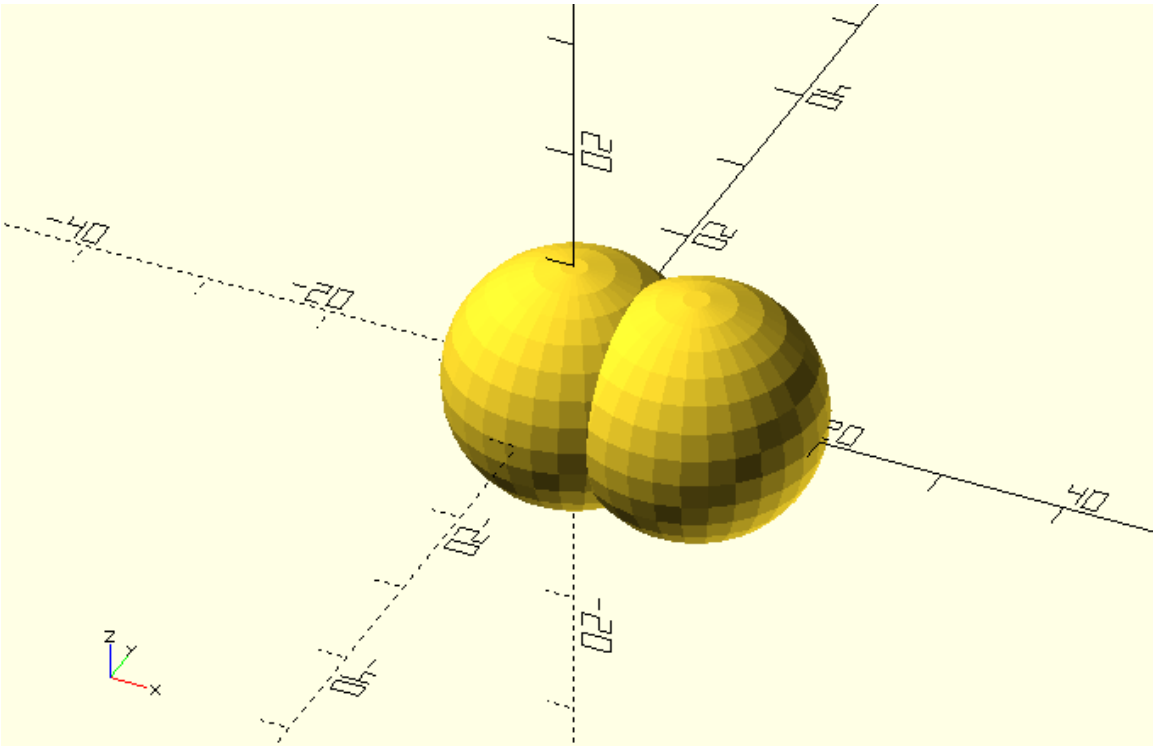
translate([20,0,0])rotate([90,0,0])cylinder(h=track,r=2,center=true);



Combining objects in other ways

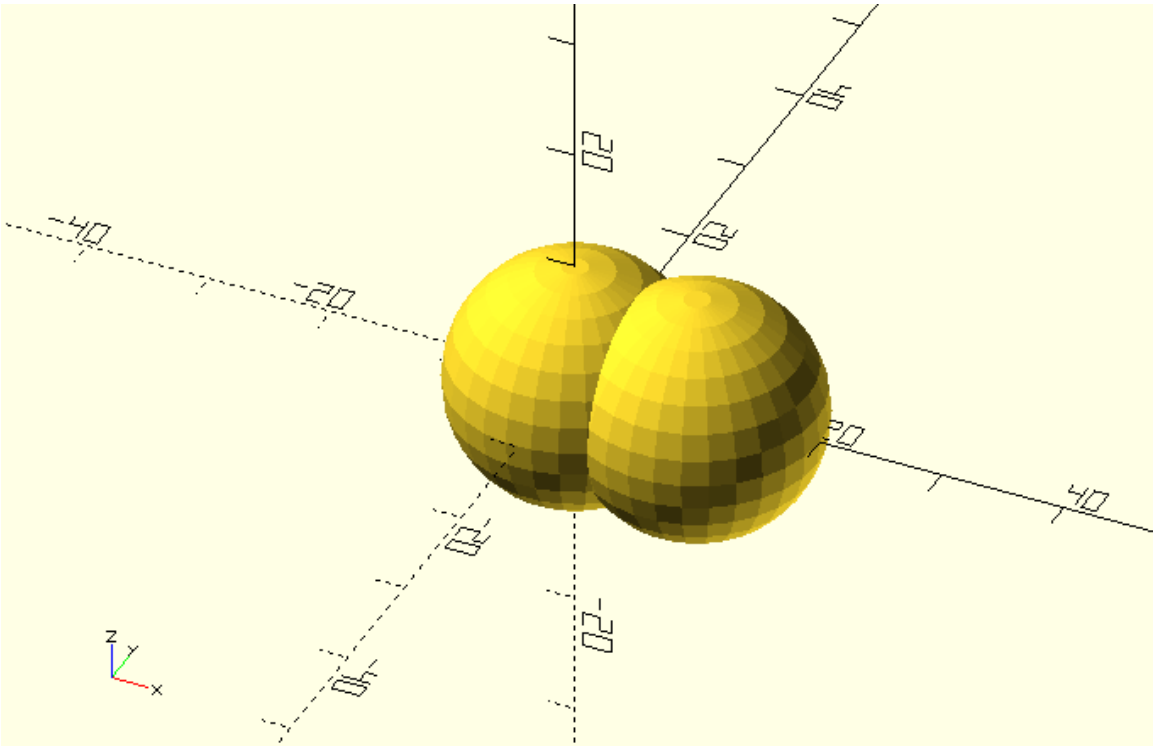
So far when you wanted to create an additional object in your model, you just added another statement in your script. The final car model is the union of all objects that have been defined in your script. You have been implicitly using the union command which is one of the available boolean operations. When using the union boolean operation, OpenSCAD takes the union of all objects as the resulting model. In the following script the union is used implicitly.

```
sphere(r=10);

translate([10,0,0])sphere(r=10);
```



You can make the use of union explicit by including the union command in your script.

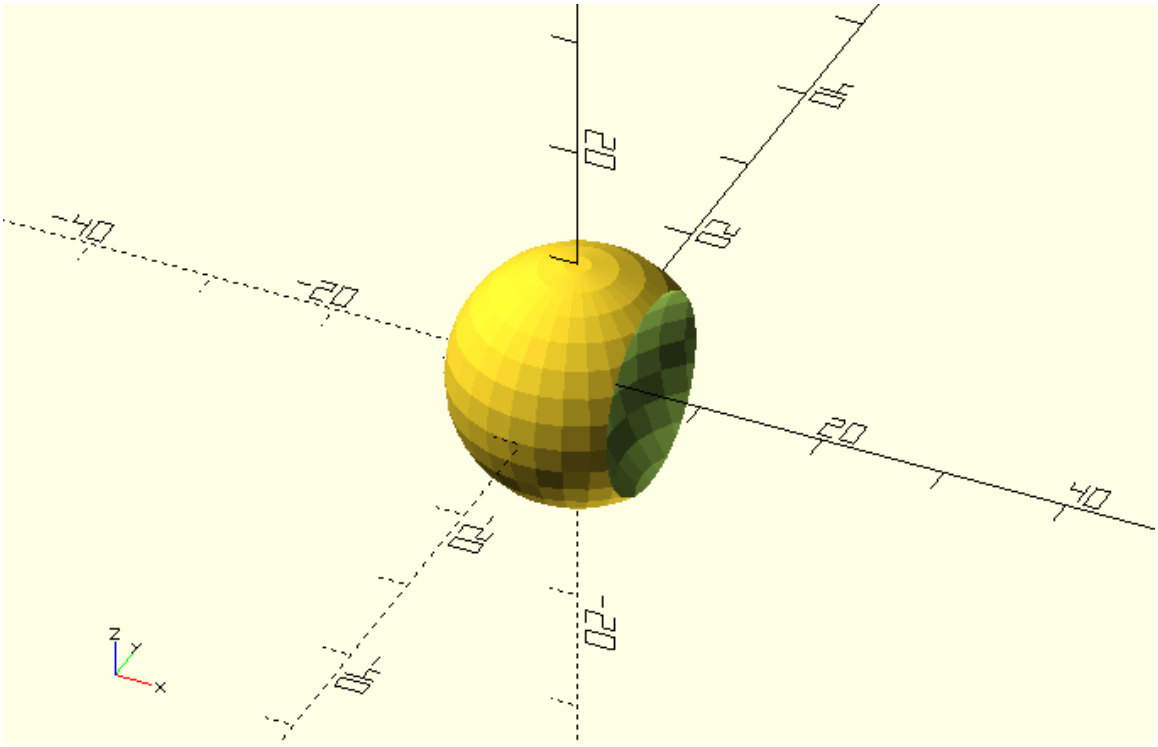
```
union(){

sphere(r=10);

translate([12,0,0])sphere(r=10);

}
```



You should notice that the union command doesn't have any input parameters. This is true for all boolean operations. The union is applied to all objects inside the curly brackets. You should also notice that the statements inside the curly brackets have a semicolon at the end. In contrast there is no semicolon after the closing curly bracket. This syntax is similar to the use of transformations when applied to multiple objects.

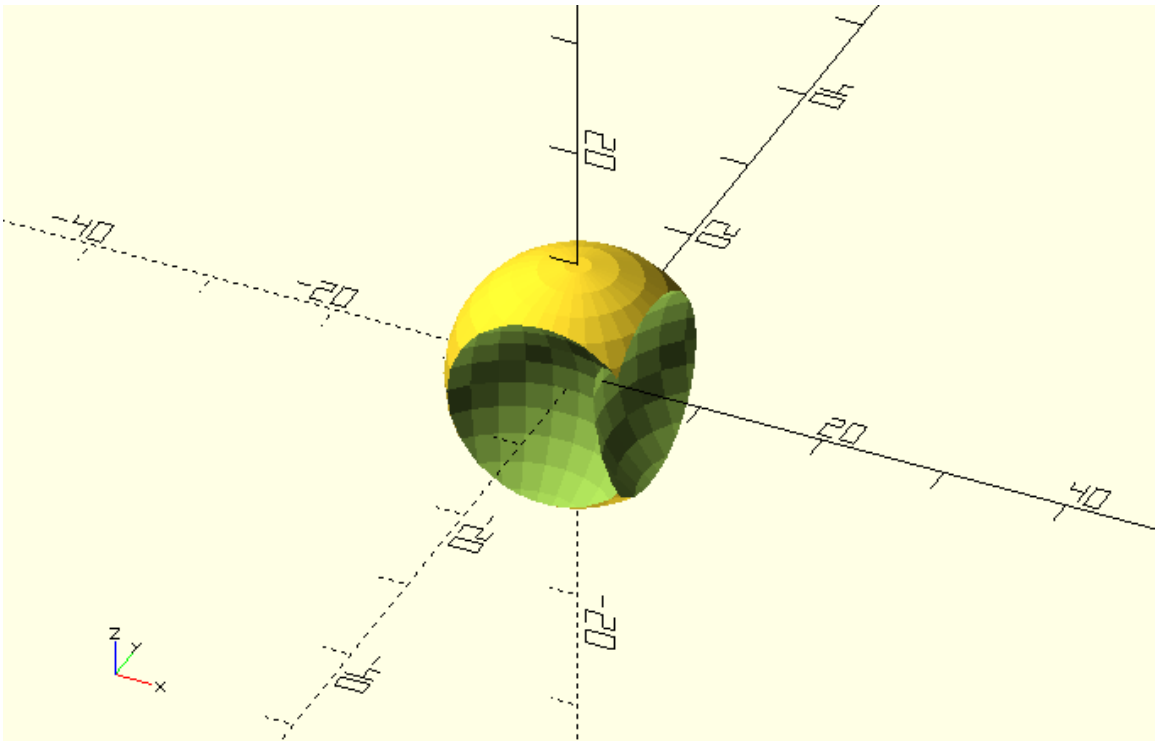
In total there are three boolean operations. The second one is the difference. The difference command subtracts the second and all further objects that have been defined inside the curly brackets from the first one. The previous example results in the following model when using the difference operation instead of the union.

```
difference(){

sphere(r=10);

translate([12,0,0])sphere(r=10);

}
```



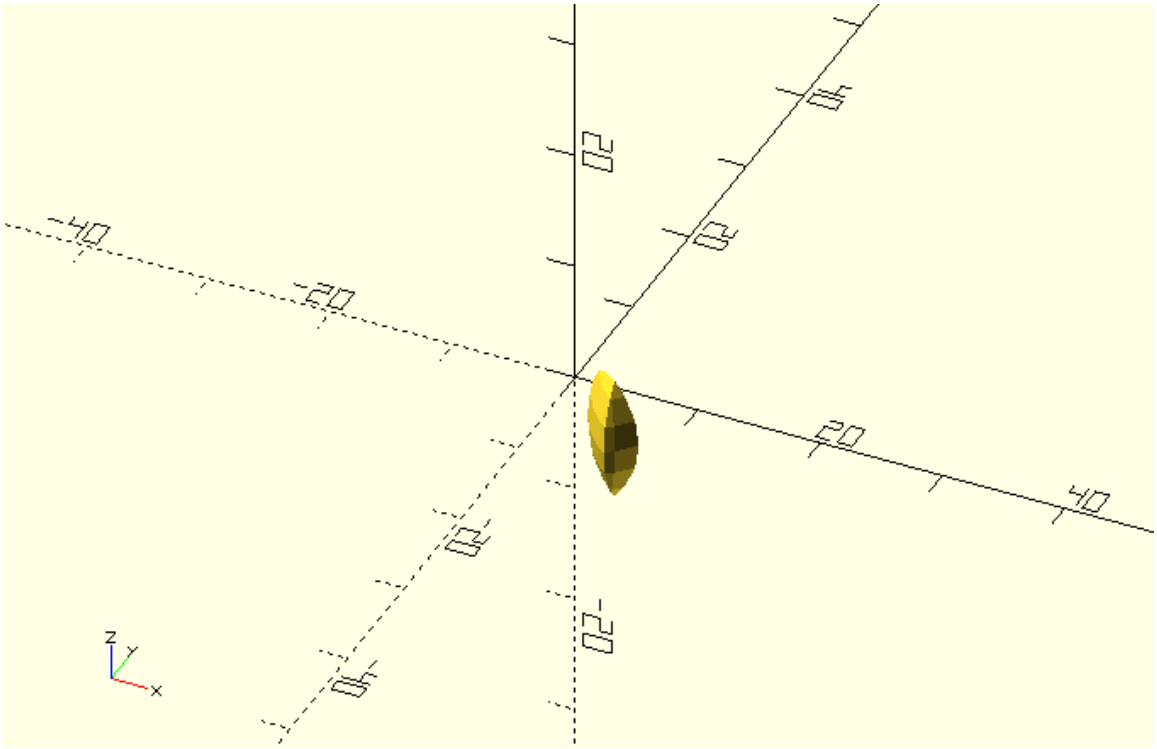
Further defined objects (third, fourth etc.) are also subtracted. The following example has three objects.

```
difference(){

sphere(r=10);

translate([12,0,0])sphere(r=10);

translate([0,-12,0])sphere(r=10);

}
```



The third boolean operation is the intersection. The intersection operation keeps only the overlapping portion of all objects. The previous example results in the following model when the intersection operation is used.

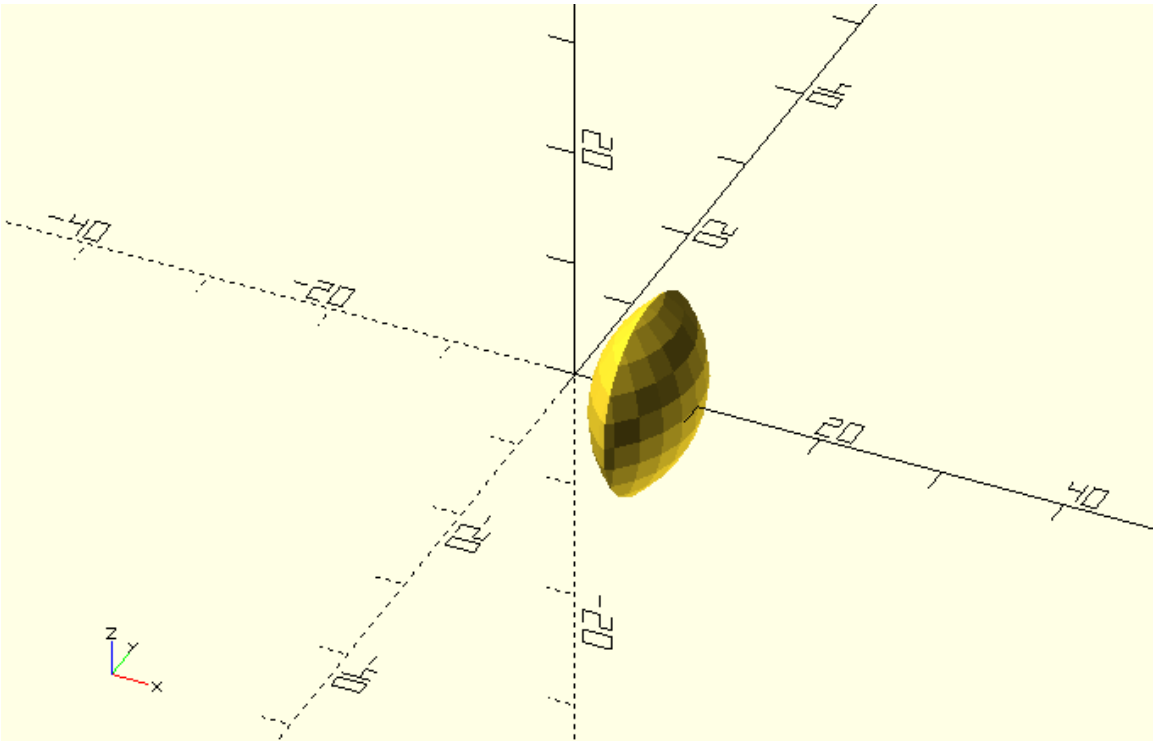
```
intersection(){

sphere(r=10);

translate([12,0,0])sphere(r=10);

translate([0,-12,0])sphere(r=10);

}
```



The resulting model is the common area of all three objects.

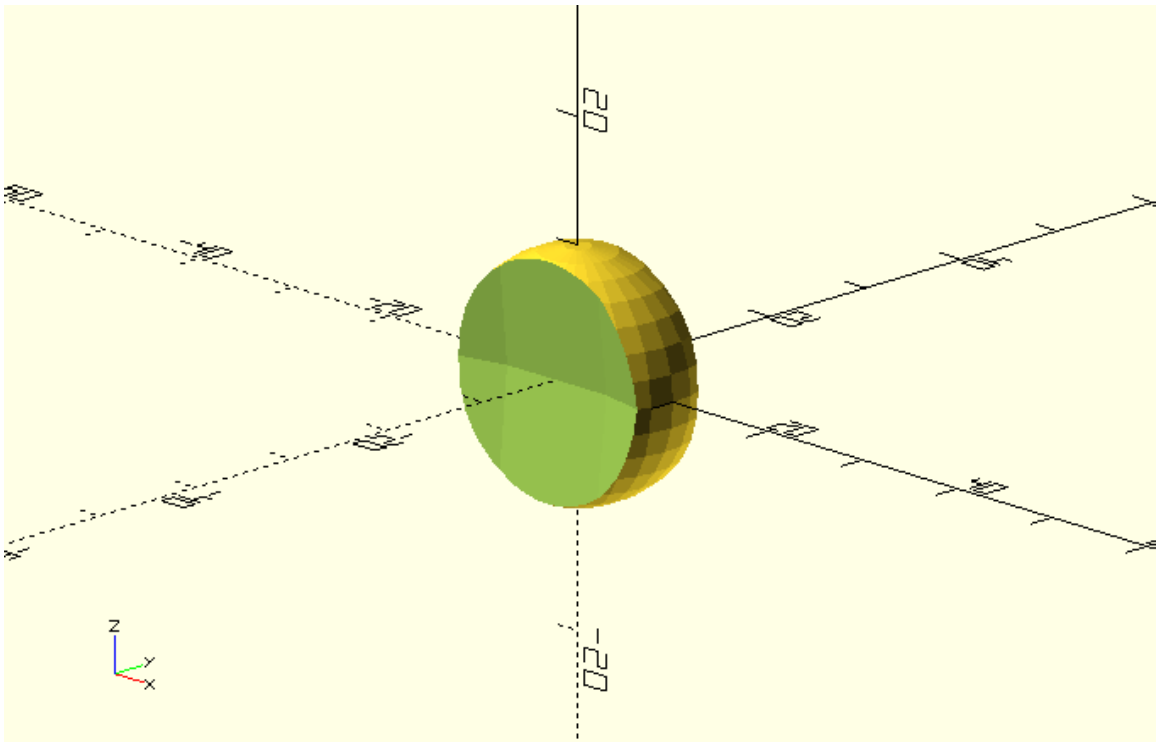
When only the first two spheres are defined inside the curly brackets, the intersection is the following.

```
intersection(){

sphere(r=10);

translate([12,0,0])sphere(r=10);

}
```



Try using the difference operation to create a new wheel design. To do so first create a sphere and then subtract a portion of a sphere from both sides. The radius of the first sphere should be equal to the desired wheel radius (wheel_radius variable). The radius of the other two spheres should be equal to a side_spheres_radius variable. Given a hub_thickness variable what is the amount of units that the side spheres should be translated along the positive and negative direction of Y axis so that the thickness of the remaining material at the center of the first sphere is equal to the value of hub_thickness?

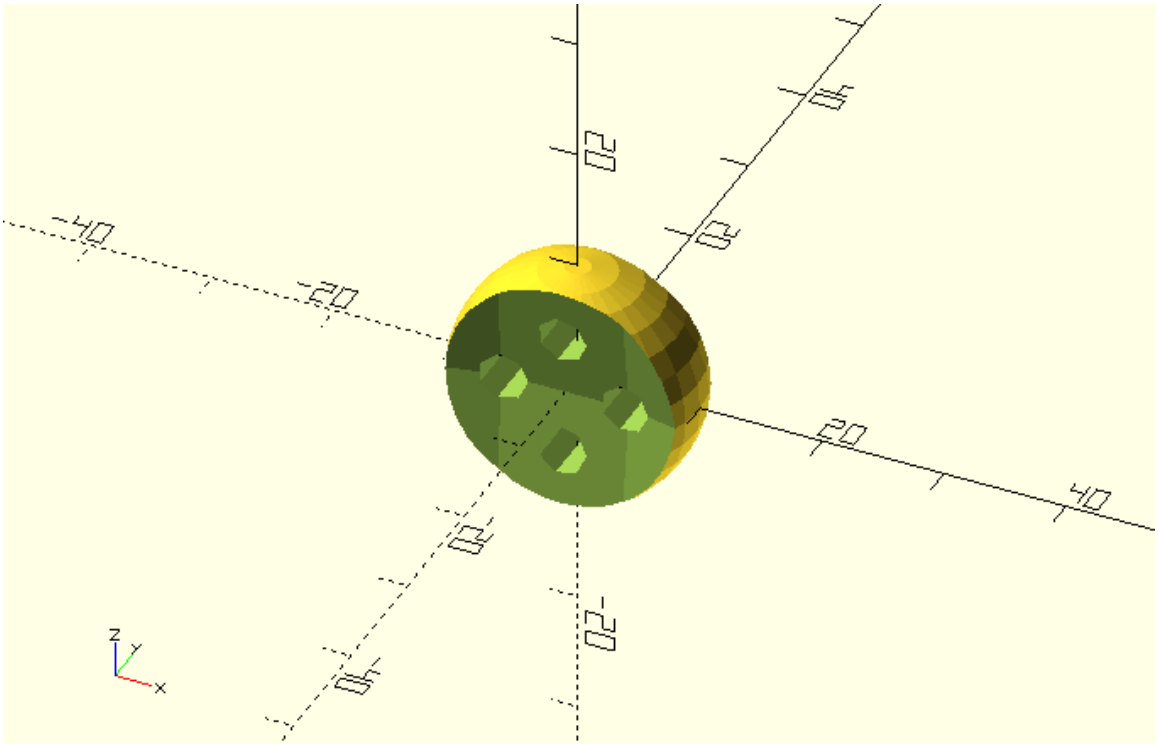
```
wheel_radius=10;

side_spheres_radius=50;

hub_thickness=4;

difference(){

sphere(r=wheel_radius);

translate([0,side_spheres_radius + hub_thickness/2,0])sphere(r=side_spheres_radius);

translate([0,- (side_spheres_radius + hub_thickness/2),0])sphere(r=side_spheres_radius);

}
```



Try removing some material from the wheels by subtracting four cylinders that are perpendicular to the wheel. The cylinders should be placed at half the wheel radius and be equally spaced. Introduce a cylinder_radius and a cylinder_height variable. The value of cylinder_height should be appropriate so that the cylinders are always longer than the thickness of the material they are removed from.

```
wheel_radius=10;

side_spheres_radius=50;

hub_thickness=4;

cylinder_radius=2;

cylinder_height=2*wheel_radius;

difference(){

// Wheel sphere

sphere(r=wheel_radius);

// Side sphere 1

translate([0,side_spheres_radius + hub_thickness/2,0])sphere(r=side_spheres_radius);

// Side sphere 2

translate([0,- (side_spheres_radius + hub_thickness/2),0])sphere(r=side_spheres_radius);
```

```
// Cylinder 1

translate([wheel_radius/2,0,0])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,center=true);

// Cylinder 2

translate([0,0,wheel_radius/2])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,center=true);

// Cylinder 3

translate([-wheel_radius/2,0,0])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,center=true);

// Cylinder 4

translate([0,0,-wheel_radius/2])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,center=true);

}
```



Try using the above wheels in one version of the car.

```
wheel_radius = 10;

base_height = 10;

top_height = 14;
```

```
track = 35;
wheel_width = 10;
body_roll = 0;
wheels_turn = 0;
side_spheres_radius=50;
hub_thickness=4;
cylinder_radius=2;
cylinder_height=2*wheel_radius;
rotate([body_roll,0,0]){
   // Car body base
   cube([60,20,base_height],center=true);
   // Car body top
translate([5,0,base_height/2+top_height/2])cube([30,20,top_height],center=true);
}
// Front left wheel
translate([-20,-track/2,0])rotate([0,0,wheels_turn])difference(){
   // Wheel sphere
   sphere(r=wheel_radius);
   // Side sphere 1
   translate([0,side_spheres_radius + hub_thickness/2,0])sphere(r=side_spheres_radius);
   // Side sphere 2
   translate([0,- (side_spheres_radius + hub_thickness/2),0])sphere(r=side_spheres_radius);
   // Cylinder 1

translate([wheel_radius/2,0,0])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,cent
er=true);
   // Cylinder 2

translate([0,0,wheel_radius/2])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,cent
er=true);
```

```
   // Cylinder 3
   translate([-
wheel_radius/2,0,0])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,center=true);
   // Cylinder 4
   translate([0,0,-
wheel_radius/2])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,center=true);
}
// Front right wheel
translate([-20,track/2,0])rotate([0,0,wheels_turn])difference(){
   // Wheel sphere
   sphere(r=wheel_radius);
   // Side sphere 1
   translate([0,side_spheres_radius + hub_thickness/2,0])sphere(r=side_spheres_radius);
   // Side sphere 2
   translate([0,- (side_spheres_radius + hub_thickness/2),0])sphere(r=side_spheres_radius);
   // Cylinder 1

translate([wheel_radius/2,0,0])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,cent
er=true);
   // Cylinder 2

translate([0,0,wheel_radius/2])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,cent
er=true);
   // Cylinder 3
   translate([-
wheel_radius/2,0,0])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,center=true);
   // Cylinder 4
   translate([0,0,-
wheel_radius/2])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,center=true);
}
// Rear left wheel
translate([20,-track/2,0])rotate([0,0,0])difference(){
```

```
   // Wheel sphere
   sphere(r=wheel_radius);
   // Side sphere 1
   translate([0,side_spheres_radius + hub_thickness/2,0])sphere(r=side_spheres_radius);
   // Side sphere 2
   translate([0,- (side_spheres_radius + hub_thickness/2),0])sphere(r=side_spheres_radius);
   // Cylinder 1

translate([wheel_radius/2,0,0])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,cent
er=true);
   // Cylinder 2

translate([0,0,wheel_radius/2])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,cent
er=true);
   // Cylinder 3
   translate([-
wheel_radius/2,0,0])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,center=true);
   // Cylinder 4
   translate([0,0,-
wheel_radius/2])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,center=true);
}
// Rear right wheel
translate([20,track/2,0])rotate([0,0,0])difference(){
   // Wheel sphere
   sphere(r=wheel_radius);
   // Side sphere 1
   translate([0,side_spheres_radius + hub_thickness/2,0])sphere(r=side_spheres_radius);
   // Side sphere 2
   translate([0,- (side_spheres_radius + hub_thickness/2),0])sphere(r=side_spheres_radius);
   // Cylinder 1
```

```
translate([wheel_radius/2,0,0])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,center=true);

   // Cylinder 2


translate([0,0,wheel_radius/2])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,center=true);

   // Cylinder 3

   translate([-wheel_radius/2,0,0])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,center=true);

   // Cylinder 4

   translate([0,0,-wheel_radius/2])rotate([90,0,0])cylinder(h=cylinder_height,r=cylinder_radius,center=true);

}

// Front axle

translate([-20,0,0])rotate([90,0,0])cylinder(h=track,r=2,center=true);

// Rear axle

translate([20,0,0])rotate([90,0,0])cylinder(h=track,r=2,center=true);
```