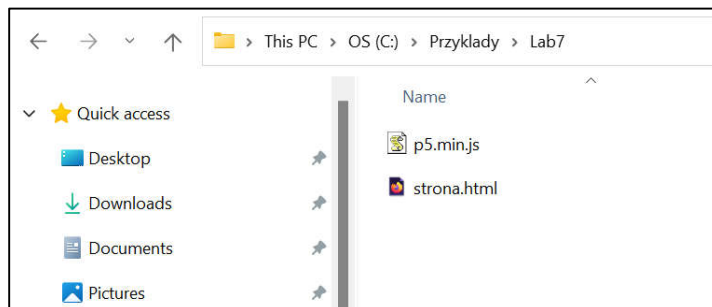


# Tworzenie Aplikacji Internetowych

## Laboratorium 6

### Wykorzystanie bibliotek JavaScript – grafika 3D na przykładzie biblioteki p5.js

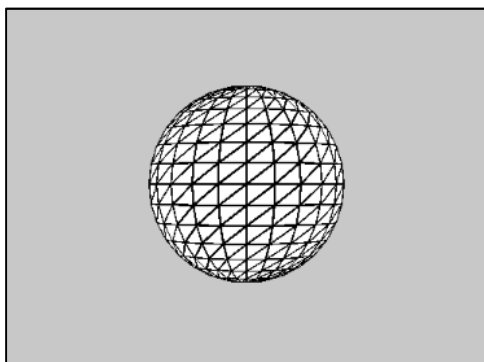
Biblioteka p5.js jest darmowa i można ją pobrać pod adresem: <https://p5js.org/> (przycisk **download** umieszczony w menu po lewej stronie). Najprostszy sposób wykorzystania biblioteki to pobranie pliku **p5.min.js** i umieszczenie go bezpośrednio w folderze tworzonej strony internetowej:



Aby wykorzystać bibliotekę należy w sekcji **head** załączyć bibliotekę **p5.min.js**, w **body** dodać znacznik **main**, oraz dodać dwie funkcje JavaScript: **setup** – funkcja ta będzie wywołana raz i powinna zainicjalizować obszar roboczy oraz **draw** – funkcja ta będzie automatycznie wywoływana około 60 razy na sekundę. Poniżej przykład który narysuje sferę w przestrzeni trójwymiarowej:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <script src="p5.min.js"></script>
  </head>
  <body>
    <main></main>
    <script>
      function setup() {
        createCanvas(640, 480, WEBGL); // inicjalizacja obszaru roboczego
      }
      function draw() {
        background(200); // wymazanie obszaru - kolor jasno-szary
        sphere(); // narysowanie sfery o domyślnym rozmiarze
      }
    </script>
  </body>
</html>
```

Wynik działania powyższego kodu (przybliżenie):

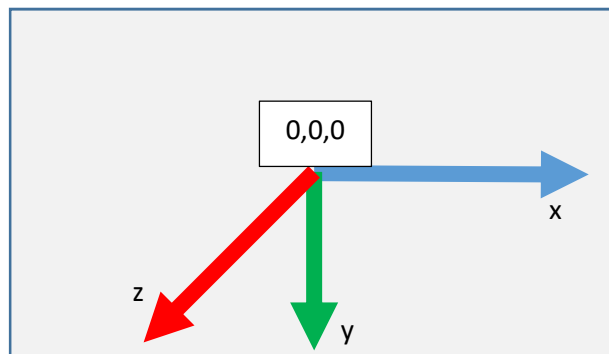


## Wykorzystanie biblioteki.

Biblioteka p5.js posiada następujące podstawowe funkcje:

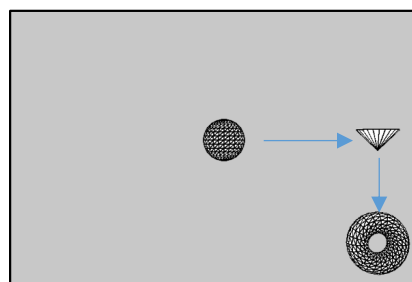
- `background(odcien_szarości)` – wymazuje obszar roboczy danym odcieniem – w zakresie od 0 do 255
- `background(r, g, b)` – j.w., ale z wykorzystaniem składowych r, g, b, parametry w zakresie od 0 do 255
- `fill(odcien_szarości)` – określenie koloru wypełnienia kształtu, parametr w zakresie od 0 do 255\*
- `fill(r, g, b)` – j.w., ale z wykorzystaniem składowych r, g, b, parametry w zakresie od 0 do 255\*
- `noFill()` – brak wypełnienia (rysowany sam szkielet obiektu 3d zbudowany z krawędzi)\*
- `stroke(odcien_szarości)` – określenie koloru linii rysowanych kształtów\*
- `stroke(r, g, b)` – j.w.\*
- `noStroke()` – brak linii obiektu 3d (rysowane samo wypełnienie kształtu)\*
- `orbitControl()` – daje możliwość obracania obszarem 3d za pomocą kursora
- `plane(szerokość, długość)` – rysuje płaszczyznę 2d
- `box(szerokość, [długość, wysokość])` – rysuje sześcian
- `cylinder(promień, wysokość)` – rysuje cylinder
- `cone(promień, wysokość)` – rysuje stożek
- `torus(promień_zewnętrzny, promień_wewnętrzny)` – rysuje torus
- `sphere(szerokość, [długość, wysokość])` – rysuje sferę

Środek układu współrzędnych znajduje się domyślnie na środku obszaru roboczego:



Obiekty trójwymiarowe są **domyślnie** rysowane w środku obszaru roboczego. Aby zmienić współrzędne należy dokonać przesunięcia układu za pomocą metody `translate(x, y, z)`. Chcąc narysować coś dalej względem osi Z należy zatem dokonać przesunięcia np.: `translate(0, 0, -100)`, chcąc narysować coś wyżej i z prawej strony: `translate(100, -100, 0)`, itd. **UWAGA: raz dokonane przesunięcie zostaje zapamiętane, i kolejne przesunięcia są do niego dodawane, pokazuje to poniższy przykład:**

```
function draw() {  
  background(200);  
  sphere(20); // narysowanie sfery w (0, 0, 0)  
  translate(150, 0, 0); // przesunięcie po X, środek: (150, 0, 0)  
  cone(20, 20); // narysowanie stożka w (150, 0, 0)  
  translate(0, 100, 0); // przesunięcie po Y, zostaje dodane, środek: (150, 100, 0)  
  torus(20, 10); // narysowanie torusa w (150, 100, 0)  
}
```

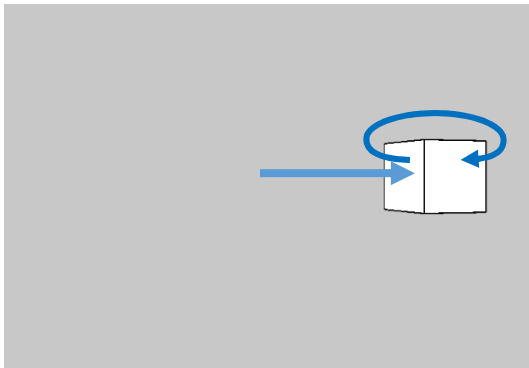


Oprócz przesunięć obiektów, można wykorzystywać transformacje wpływające na wielkość kształtu: scale(współczynnik) oraz transformacje obracające obiekty wzdłuż osi: rotateX(kąt), rotateY(kąt), rotateZ(kąt).

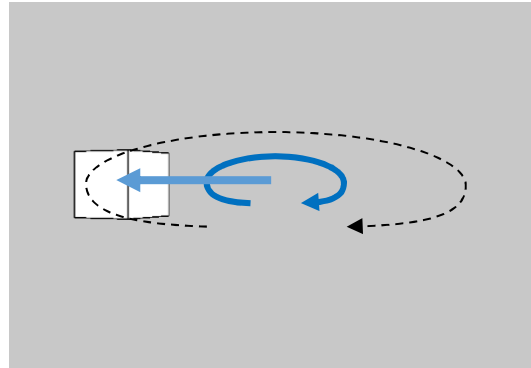
**UWAGA:** takie transformacje również zostają zapamiętane, a także wpływają na wywoływane później translate (środek współrzędnych jest przesuwany o dane współrzędne uwzględniając obrót).

Kolejność przesunięć i obrotów ma zatem znaczenie, pokazują to następujące przykłady:

```
let kat = 0.0; // obiekt będzie animowany
function draw() {
  background(200);
  translate(150, 0, 0); // najpierw przesunięcie
  rotateY(kat); // potem obrót wzdłuż osi Y
  box(); // obiekt obraca się w miejscu
  kat += 0.03; // zwiększenie kąta obrotu
}
```



```
let kat = 0.0; // obiekt będzie animowany
function draw() {
  background(200);
  rotateY(kat); // najpierw obrót
  translate(150, 0, 0); // przesunięcie z obrotem
  box(); // obraca się wokół (0, 0, 0)
  kat += 0.03; // zwiększenie kąta obrotu
}
```



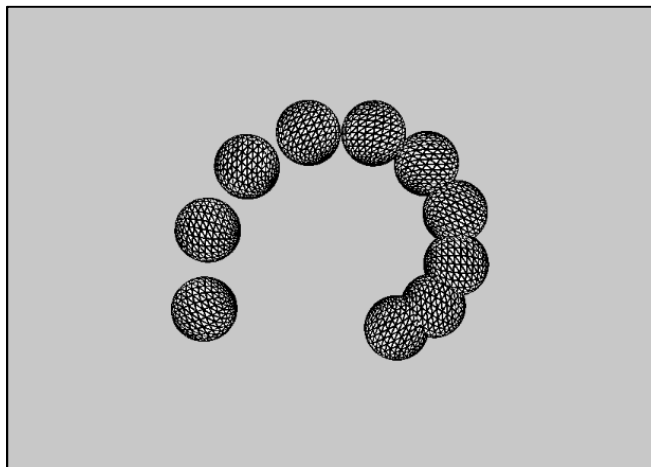
Ponieważ wszystkie transformacje są zapamiętywane, do pracy z nimi dodano dwie metody:

- push(); // zapamiętuje transformacje na stosie
- pop(); // zdejmuję transformację ze stosu

Dzięki temu można w łatwy sposób powrócić do środka układu współrzędnych:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <script src="p5.min.js"></script>
  </head>
  <body>
    <main></main>
    <script>
      function setup() {
        createCanvas(640, 480, WEBGL);
      }
      let kat = 0.0; // obiekt będzie animowany
      function draw() {
        background(200);
        for (let i = 0; i < 10; i++) { // narysowanie 10 sfer
          push(); // zapamiętanie transformacji
          rotate(kat * (i + 1)); // każda sfera będzie obrócona o inny kąt
          translate(0, 50 + kat * i, 0); // i przesunięta dalej po osi Y
          sphere(20); // narysowanie sfery
          pop(); // obrót i przesunięcie zostaną zniwelowane dla kolejnej sfery
        }
        kat += 0.01; // zwiększenie kąta obrotu
      }
    </script>
  </body>
</html>
```

Wynik powyższego skryptu:



Aby obiekt obracał się jednocześnie wzdłuż danej osi i wokół swojego środka należy użyć obrotu dwukrotnie: `rotateY(kat); translate(tx, 0, 0); rotateY(kat)`. Natomiast aby obiekt obracał się tylko wzdłuż danej osi i był zawsze ułożony w taki sam sposób względem kamery należy odwrócić go po przesunięciu o ujemną wartość ustawianego wcześniej kąta: `rotateY(kat); translate(tx, 0, 0); rotateY(-kat)`.

### Zadanie 1

- Pobrać bibliotekę **p5.js** i umieścić obok tworzonej strony internetowej (tak jak na stronie pierwszej)
- Przetestować przykład ze strony trzeciej niniejszej instrukcji
- Zwiększyć liczbę sfer do 20 (nie sprawdzać dla 200, a już z pewnością dla 2000 sfer)
- Zakomentować funkcję `pop()` i sprawdzić efekt, następnie z powrotem odkomentować funkcję `pop()`
- Dodać transformację skalującą każdą sferę o wartość kąta dzieloną przez 5.

### Zadanie 2

- Bazując na poprzednim zadaniu usunąć pętlę rysującą sfery (zostawić zmienną `kat` i jej zwiększanie).
- Dodać pętlę, która narysuje pięć sześciątów o współrzędnych `-200;0;0, -100;0;0, 0;0;0, 100;0;0, 200;0;0` - czyli dokonywać przesunięć po osi X o konkretną wartość (pamiętać o `push()` oraz `pop()`)
- Po wywołaniu metody przesunięcia wywoływać metodę obrotu wzdłuż osi X o wartość zmiennej `kat`.
- Dodać warunek, który dodatkowo obróci środkowy sześciąt wzdłuż osi Z o wartość zmiennej `kat`.
- Zagnieżdżyć utworzoną pętlę w kolejną, tak aby pętla zewnętrzna pozwoliła na dokonywanie przesunięć po osi Y o kolejne wartości `-200; -100; 0; 100; 200`. Wynikiem działania zagnieżdżonych pętli powinno być narysowanie **25 sześciątów (5 wierszy po 5 sześciątów w rzędzie)**.
- Pod koniec funkcji rysującej (`draw`) wywołać metodę `orbitControl()`; Sprawdzić możliwość obrotu sceny za pomocą myszki.

### Światła i materiały

**P5.js** umożliwia dodawanie światel i nadawanie materiałów obiektom 3D. Światła należy dodać w funkcji `draw` przed rysowaniem obiektów. Materiały można nadawać obiektom bezpośrednio przed ich narysowaniem:

- `lights()` – uruchomienie światła typu ambient oraz kierunkowego o domyślnych parametrach
- `pointLight(r, g, b, x, y, z)` – punkt światła o danym kolorze oraz współrzędnych
- `directionalLight(r, g, b, x, y, z)` – j.w ale źródło światła jest ustawione w konkretnym kierunku
- `spotLight(r, g, b, x, y, z, kx, ky, kz)` – światło kierunkowe
- `ambientLight(r, g, b)` – ogólne światło bez konkretnego położenia
- `fill(r, g, b)` - standardowy materiał

- texture(obrazek) – wypełnienie teksturą
- ambientMaterial(r, g, b) – określenie ile dany kolor odbija światła ambient
- specularMaterial(r, g, b) – nadanie materiału z odbiciem światła (dodatkowa metoda: shininess(poziom) - określa poziom odbicia światła)
- normalMaterial() – brak odbicia światła – kolor według kierunku wektora normalnego

**UWAGA:** obrazki służące za tekstury można wczytywać za pomocą funkcji loadImage(ścieżka). Możliwe jest wykorzystanie tylko plików z serwerów WWW (także serwera lokalnego). Wczytywanie obrazków z dysku nie jest możliwe. To samo dotyczy wczytywania obiektów 3D (metoda loadModel(ścieżka)).

**UWAGA:** nie każdy serwer daje prawo do wczytania obrazków – część porównanych adresów url może nie działać

Obrazki najlepiej wczytywać w dodatkowej metodzie preload i umieszczać w zmiennych globalnych. Metoda ta jest wykonywana przed załadowaniem sceny. Pokazuje to następujący przykład:

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <script src="p5.min.js"></script>
    <style>
      main { position: fixed; left: 0; top: 0; z-index: -1; }
      div div { margin: 100px; padding: 10px; width: 200px; height: 100px;
        border-radius: 10px; background: white; opacity: 0.8; }
    </style>
  </head>
  <body>
    <main></main>
    <script>
      let rysunek = null;
      function preload() { // wczytanie tekstury wewnątrz tej metody
        rysunek = loadImage("https://cdn.pixabay.com/photo/2018/05/17/15/24/armillar-ball-3408811_1280.jpg");
      }
      function setup() {
        createCanvas(windowWidth, windowHeight, WEBGL); // obszar roboczy na całe okno
        noStroke();
      }
      function windowResized() {
        resizeCanvas(windowWidth, windowHeight); // zmiana przy zmianie wielkości okna
      }
      let kat = 0.0;
      function draw() {
        background(0, 10, 40); // kolor tła - ciemno niebieski
        lights(); // włączanie podstawowych świateł
        rotateY(kat); // obrót obiektu
        texture(rysunek); // wybranie tekstury na materiał obiektu
        sphere(100); // narysowanie sfery
        kat += 0.02;
      }
    </script>
  </body>
</html>
```

W powyższym przykładzie wykorzystane są zmienne widthWidth oraz windowHeight, pod którymi tak naprawdę zapisana jest wielkość obszaru strony internetowej (window.innerWidth i windowHeight). Dodatkowo p5.js umożliwia wykorzystanie zmiennych i metod:

- mouseX – pozycja X kursora nad rysowanym obszarem
- mouseY – pozycja Y kursora nad rysowanym obszarem

- $\sin(\text{kat})$ ,  $\cos(\text{kat})$ , ... - funkcje matematyczne (normalnie należało użyć: `Math.sin(kat)`, ...)
- `random(wartość)` – liczba losowa od 0 do danej wartości
- `keyPressed()` – po utworzeniu tej funkcji będzie wywoływana przy wciśnięciu klawisza
- `keyReleased()` – po utworzeniu tej funkcji będzie wywoływana przy puszczeniu klawisze
- `keyCode` – kod wciśniętego klawisza (np. `LEFT_ARROW`, `RIGHT_ARROW`)

## Kamera

Kamerę można ustawiać albo wywołując metodę `camera(px, py, pz, cx, cy, cz)`, gdzie `px, py` i `pz` określają położenie kamery, `cx, cy, cz` określają miejsce patrzenia kamery. Albo tworząc zmienną globalną dla kamery `cam = createCamera()`; i wywołując dla tej zmiennej metody, np.:

- `.ortho(...)` – widok ortogonalny
- `.pan(...)` – obrót kamery w lewo i prawo
- `.tilt(...)` – obrót kamery w górę i w dół
- `.move(...)` – przesunięcie kamery
- `.lookAt(...)` – zmiana współrzędnych na które patrzy kamera

## Rysowanie kształtów

Aby rozpocząć tworzenie dowolnych kształtów należy wywołać metodę `beginShape(parametr)`, parametr:

- `brak` – kształt określony przez dane punkty
- `POINTS` – narysowanie tylko punktów o danych współrzędnych
- `LINES` – rysowanie linii pomiędzy kolejnymi dwoma współrzędnymi
- `TRIANGLES` – rysowanie trójkątów między kolejnymi trzema współrzędnymi
- `TRIANGLE_STIP` – trójkąty są tworzone dla każdego z trzech umieszczonych obok siebie punktów

Następnie należy podawać kolejne współrzędne punktów za pomocą `vertex(x, y, z)` i zakończyć kształt za pomocą metody `endShape([CLOSE])`.

Więcej informacji oraz opisanych funkcjonalności można znaleźć na stronie: <https://p5js.org/get-started/>

## Zadanie 3

- Uruchomić przykład z piątej strony tej instrukcji.
- Spróbować zastąpić przykładową teksturę inną teksturą (tekstury ze strony <https://pixabay.com/pl/> powinny działać prawidłowo – należy podać bezpośredni linki do obrazków)
- **(opcjonalne)** Za pomocą pętli dodać na scenie kilka sfer okrążających sferę z przykładu (należy przed każdym narysowaniem sfery w pętli obrócić ją o inny kąt dla każdej sfery po osi X, Y oraz Z, a następnie wykonać `translate` w każdej osi (czyli np. `translate(150, 150, 150)`). Przykład transformacji przed narysowaniem sfery, gdzie: `i` to zmienna z pętli tworzącej sfery (przykład uproszczony i nierealistyczny):

```
rotateX(kat * i);
rotateY(kat * i / 3);
rotateZ(kat / i)
translate(150, 150, 150);
```

- Pod sam koniec funkcji rysującej (`draw`) wywołać metodę zmieniającą położenie kamery: `camera` podając za współrzędne położenia kamery wartości `(0, mouseX, 500)`, a za miejsce patrzenia kamery współrzędne `(0, 0, 0)`. Sprawdzić czy pozycja kamery zmienia się w zależności od położenia kursora.
- Na stronie **pod** znacznikiem `main`. Umieścić `div`, wewnątrz którego umieścić kilka ułożonych jeden pod drugim `div`ów i wpisując do ich środka dowolną treść. Umieścić tyle elementów `div` aby pojawił się pionowy pasek przewijania strony. Sprawdzić zachowanie się strony.
- Zmodyfikować zmianę pozycji kamery, w taki sposób aby pozycja kamery i punkt patrzenia kamery zależała także w jakiś dowolny sposób od pionowego paska przewijania strony (tzn. wykorzystać w parametrach metody `camera` zmienną `window.scrollY` lub `window.scrollY/2`). Zmienną `window.scrollY` wstawiać w różnych miejscach jako parametr funkcji `camera` do uzyskania ciekawego rezultatu.

#### Zadanie 4 (opcjonalne)

- Bazując na zadaniu 2, zmodyfikować je w taki sposób, aby obliczać odległość od środka każdego sześcianu (pozycje zgodne z przesunięciami) do pozycji kursora (zmienne `mouseX` i `mouseY`). Należy pamiętać że środek układu współrzędnych znajduje się w pozycji kursora równej  $\text{width}/2$  i  $\text{height}/2$ . Następnie zależnie od tej odległości odpowiednio przeskalować każdy sześcian (przed jego narysowaniem). Warto obliczoną odległość podzielić przez około 500 przed ustawieniem skalowania.
- Zastąpić sześciany dowolnym innym wybranym kształtem i spróbować nadać każdemu kształtowi inny kolor (zależny od jego pozycji w przestrzeni).

#### Zadanie 5 (opcjonalne)

- Bazując na zadaniu 2, zmodyfikować je w taki sposób, aby nie wywoływać ani `push()` oraz `pop()`.