VI. МОДЕЛИРОВАНИЕ ДИНАМИЧЕСКИХ 3D ПРОЦЕССОВ

Моделирование динамических 3D процессов, помимо научных целей, может иметь и прикладную ценность.

Графическая библиотека p5.js может быть использована для физического моделирования динамических процессов. Это позволяет моделировать механические системы (в рамках законов теоретической механики). С помощью p5.js можно моделировать поступательные и вращательные движения в трех плоскостях.

В примере, показанном на рисунке 6.1, моделируется столкновение двух сфер с массами *m*1 и *m*2, которые до момента столкновения имеют скорости линейного смещения центров, обозначенных *v*1 и *v*2. После столкновения сферы будут двигаться со скоростью *v*'1 и *v*'2. Сферы катятся по жесткой поверхности, то есть совершают вращательное движение, заданное углами: « omega1» и « omega2».

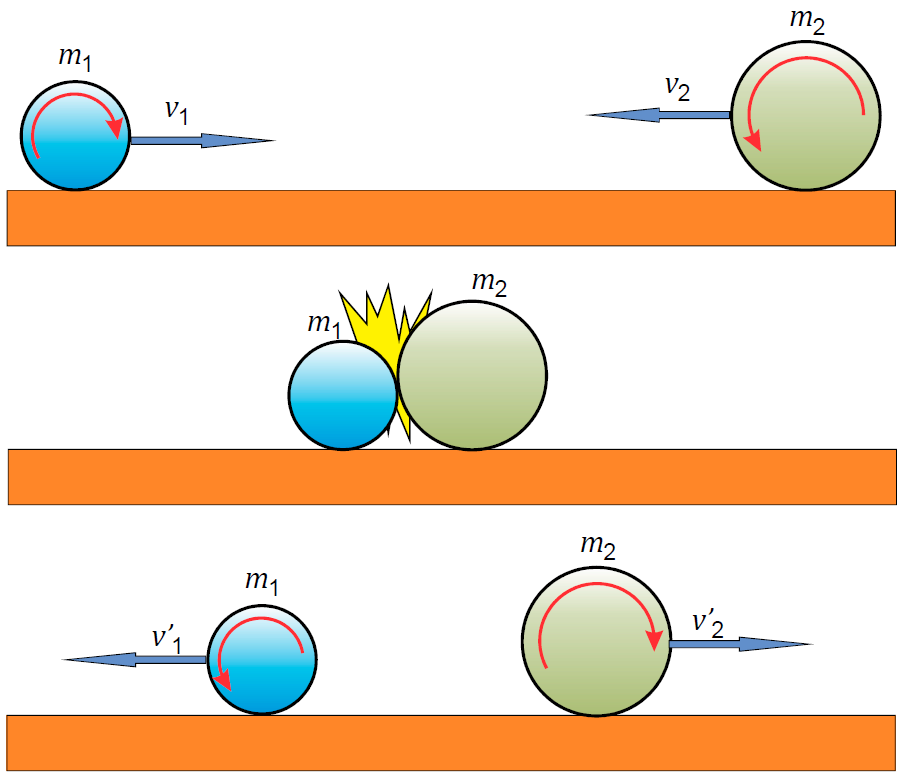


Рисунок 6.1. Столкновение двух сфер

Используя закон сохранения механической энергии, формулу (6.1):

, (6.1)

и закон сохранения импульса, формулу (6.2):

, (6.2)

можно определить соотношения между скоростями до и после столкновения сфер (формулы 6.3 и 6.4):

, (6.3)

. (6.4)

Угол поворота можно выразить с помощью формул 6.5 и 6.6.

(6.5)

(6.6)

Модель этого эксперимента может быть описана следующим кодом:

**Программа:**

let x1=-150;

let v1=5;

let m1=50;

let omega1=v1/m1;

let a1=0;

let x2=150;

let v2=-15;

let m2=20;

let omega2=v2/m2;

let a2=0;

function setup()

{ createCanvas(500, 500, WEBGL);

strokeWeight(0.2);

frameRate(24); }

function draw()

{ background(200);

orbitControl();

pointLight(255, 255, 255, 0, 0, 1000);

push();

box(500,100,100);

pop();

push();

translate(x1, -100/2-m1, 0);

rotateZ(a1);

sphere(m1);

pop();

push();

translate(x2, -100/2-m2, 0);

rotateZ(a2);

sphere(m2);

pop();

if(x1+m1>x2-m2)

{ v1=2\*(m1\*v1+m2\*v2)/(m1+m1)-v1;

v2=2\*(m2\*v2+m1\*v1)/(m1+m1)-v2;

x1=x1+v1;

x2=x2+v2;

omega1=v1/m1;

omega2=v2/m2; }

x1=x1+v1;

x2=x2+v2;

a1=a1+omega1;

a2=a2+omega2; }

**Описание программы:**

**let** – создает и называет новую переменную. Переменная – это контейнер для хранения значения.

Переменные, объявленные с помощью **let,** будут иметь область видимости блока. Это значит, что переменная существует только в том блоке, в котором она создана.

В вышеуказанной программе в качестве переменных рассматриваем:

– х1 и х2 – начальные положения сфер;

– v1 и v2 – начальные линейные скорости центров сфер;

– масса шаров m1 и m2, которые также выражают радиусы сфер;

– omega1=v1/м1 и omega2=v2/м2 – углы поворота сфер;

– a1 = 0 и a2 = 0 – начальные углы поворота сфер.

Поверхность, по которой катятся сферы, построена с помощью функции box**(500,100,100),** являясь статичным объектом.

Сферы рисуются с помощью функции **sphere(),** к ним применяется поступательное движение с помощью функции **translate()** и вращательное движение с помощью **функции rotateZ().**

Модели трехмерных объектов вызываются в каждом отдельном блоке, разделяемом push **()**, **pop()**.

Последняя часть программы включает в себя формулы расчета скоростей сфер, их положения и угла поворота.

Результат выполнения программы представлен на рисунке 6.2.

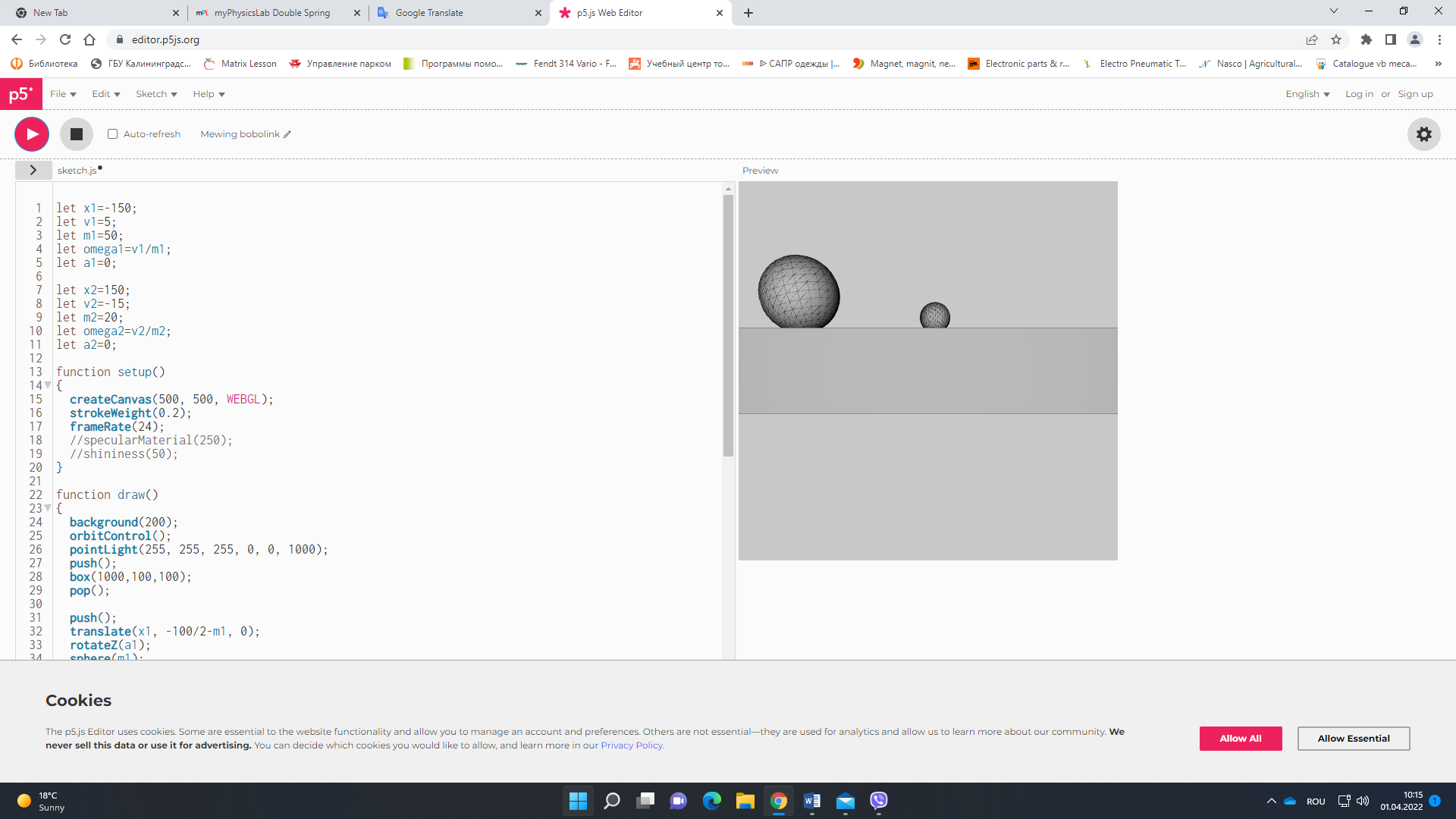


Рисунок 6.2. Модель симуляции столкновения 2-х сфер в p5.js

Для создания модели используются 3D-примитивы из библиотеки p5.js.

### 6.1. Создание простых 3D-графических примитивов

Простые 3D-графические примитивы, такие как следующие, используются для создания фигур в p5.js:

* [**plane()**](https://p5js.org/reference/#/p5/plane);
* [**box()**](https://p5js.org/reference/#/p5/box);
* [**sphere()**](https://p5js.org/reference/#/p5/sphere);
* [**cylinder()**](https://p5js.org/reference/#/p5/cylinder);
* [**cone()**](https://p5js.org/reference/#/p5/cone);
* [**ellipsoid()**](https://p5js.org/reference/#/p5/ellipsoid);
* [**torus()**](https://p5js.org/reference/#/p5/torus).

**Функция plane():** рисует плоскость с заданной шириной и высотой.

**Синтаксис:**

**plane([width],[height], [detailX], [detailY]);**

где: [width]: ширина (опционально);

[height]: высота (опционально);

detailX: количество треугольников по оси x (опционально);

detailY: количество треугольников по оси y (опционально);

**Функция box():** рисует параллелепипед с указанной шириной, высотой и глубиной.

**Синтаксис:**

**box([width],[Height],[depth],[detailX], [detailY]);**

где: [width]: ширина фигуры (опционально);

[height]: высота фигуры (опционально);

[depth]: глубина фигуры (опционально);

detailX: количество треугольников по оси x (опционально);

detailY: количество треугольников по оси y (опционально).

**Функция sphere():** рисует сферу с указанным радиусом. DetailX и detailY определяют количество отрезков в x-измерении и y-измерении сферы. Большее количество отрезков делает сферу более гладкой. Максимальное рекомендуемое значение – 24. Использование значения выше 24 может замедлить работу браузера.

**Синтаксис:**

**sphere([radius], [detailX], [detailY]);**

где: radius: радиус сферы (опционально);

detailX: количество отрезков по оси x (опционально);

detailY: количество отрезков по оси y (опционально);

**Функция cylinder():** рисует цилиндр с заданным радиусом и высотой. detailX и detailY определяют количество отрезков по осям x и y цилиндра. Большее количество отрезков делает цилиндр более гладким. Максимальное рекомендуемое значение для detailX равно 24. Использование значения выше 24 может замедлить работу браузера.

**Синтаксис:**

**cylinder([radius], [height], [detailX], [detailY], [bottomCap], [topCap]);**

где: radius: радиус основания цилиндра (опционально);

height: высота цилиндра (опционально);

detailX: количество отрезков по оси x, по умолчанию 24 (опционально);

detailY: количество отрезков по оси y, по умолчанию 1 (необязательно);

bottomCap Boolean: рисует или не рисует дно цилиндра (опционально)

topBoolean head: рисует или не рисует верхнюю часть цилиндра (опционально).

**Функция cone():** рисует конус с радиусом и заданной высотой. DetailX и detailY определяют количество отрезков по осям x и y конуса. Несколько отрезков делают конус более гладким. Максимальное рекомендуемое значение для detailX равно 24. Использование значения выше 24 может замедлить работу браузера.

**Синтаксис:**

**cone([radius], [height], [detailX], [detailY], [cap]);**

где: radius: радиус основания конуса (опционально);

height: высота конуса (опционально);

detailX: количество сегментов по оси x, по умолчанию 24 (опционально);

detailY: количество сегментов по оси y, по умолчанию 1 (опционально);

Cap Boolean: рисует или не рисует основание конуса (опционально).

**Функция ellipsoid():** рисует эллипсоид. DetailX и detailY определяют количество сегментов по осям x и y рисунка. Большее количество сегментов делает эллипсоид более плавным. Избегайте детализации более 150, так как это может привести к сбою вашего браузера.

**Синтаксис:**

**ellipsoid([radiusx], [radiusy], [radiusz], [detailX], [detailY]);**

где: radiusx: x - радиус эллипсоида (опционально);

radiusy: y - радиус эллипсоида (опционально);

radiusz: z- радиус эллипсоида (опционально);

detailX: количество отрезков по оси X, по умолчанию 24. Избегание количества отрезков больше 150 может заблокировать браузер. (опционально);

detailY: количество отрезков по оси y, по умолчанию 16. Выше 150 может заблокировать браузер. (опционально).

**Функция torus():** рисует тор (бублик). DetalX и detalY определяют количество сегментов по оси x и оси y тора. Максимальные значения по умолчанию для detailX и detailY составляют 24 и 16 соответственно. Установка для них относительно низких значений, таких как 4 и 6, позволяет создавать новые формы, отличные от тора.

**Синтаксис:**

**torus([radius], [tubeRadius], [detailX], [detailY]);**

**Параметры:**

radius: внешний радиус фигуры (опционально);

tubeRadius: внутренний радиус фигуры (опционально);

detailX: количество отрезков по оси X, по умолчанию 24 (опционально);

detailY: количество отрезков по оси y, по умолчанию 16 (опционально).

### Лабораторные работы No 6

### Тема: МОДЕЛИРОВАНИЕ ДИНАМИЧЕСКИХ 3D ПРОЦЕССОВ

**Задачи работы:**

**1. Ознакомление с физическими процессами,** используемыми для создания 3D графических сцен.

**2. Использование графических примитивов** – получение практических знаний по использованию 3D графических примитивов для создания сцен.

**3. Создание динамических 3D графических сцен** – развитие навыков создания и компоновки динамических 3D графических сцен путем эффективного сочетания геометрических преобразований.

**4. Понимание и применение концепций физических процессов** – использование понятий скорости, мощности, гравитации и других, используемых для направления визуальной динамики в 3D-графике.

**Количество часов, необходимое для выполнения – 4 академических часа.**

**Цель работы:** получить практические знания по созданию 3D сцен при моделировании динамических 3D процессов с использованием стандартных геометрических трансформационных функций, таких как перемещение, масштабирование и вращение из библиотеки p5.js.

**Задача:** разработать программу, создающую 3D-сцену для моделирования физического процесса, используя стандартные функции переноса и вращения в библиотеке p5.js, согласно вариантам приведённых в таблице 6.1. Для создания сцены можно использовать существующие 3D-графические объекты из 3D-хранилищь.

Симуляцию и описание двумерного физического процесса для задания можно найти на странице <https://www.myphysicslab.com/>.

**Таблица 6.1. Варианты для лабораторной работы**

| Вариант | Рисунок | Вариант | Рисунок |
| --- | --- | --- | --- |
| 1. Одинарная пружина растяжения |  | 6. Жесткий маятник с движением по кругу |  |
| 2. Двойная пружина растяжения |  | 7. Двойной жесткий маятник |  |
| 3. Хаотический маятник |  | 8. Столкновение жестких блоков |  |
| 4. Движение сферы по горбообразной траектории |  | 9. Маятник Ньютона |  |
| 5. Движение сферы по нескольким траекториям |  | 10. Притяжение тел |  |

**Критерии оценки**

**1. Корректность кода** (20%) – проверка корректности кода, без синтаксических и операционных ошибок.

**2. Использование графических 3D примитивов** (20%) – оценка правильности использования и разнообразие графических примитивов в соответствии с требованиями работы.

**3. Соответствие инструкциям и требованиям** (10%) – проверка правильности выполнения требований задания, таких как определенные формы или размеры примитивов и их динамика.

**4. Оптимизация кода** (10%) – оценка эффективности кода в использовании ресурсов и избегание избыточного кода.

**5. Интерактивность** (10%) – ответы мыши или клавиатуры, будет оцениваться, правильно ли реализованы эти функции.

**6. Визуальная эстетика** (10%) – анализ визуальной привлекательности композиции, такой как цветовая гармония, пропорции и графический баланс.

**7. Соблюдение срока сдачи** (10%) – оценка балла по пунктуальности, если работа была сдана в установленный срок.

**8. Оценка знаний** (10%) – пояснения к процессу выполнения работы, которые могут включать описание основных функций и используемой логики.

**Вопросы для проверки знаний**

1. Перечислите простые примитивы 3D-графики.

2. Перечислите функции, используемые для графических преобразований.

3. Как можно изменить атрибуты отображения графических 3D-примитивов?

4. Виды вращения и соответствующие функции.

5. Какие графические примитивы можно использовать для создания базовых фигур в 2D-графической сцене?

6. Как можно добиться комбинации графических преобразований?

7. Какую роль играют push() и pop() в применении геометрических преобразований?

8. Как включить 3D режим в p5.js?