МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ВОРОНЕЖСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Математический факультет

Кафедра теории функций и геометрии

Программная реализация (на языке JavaScript) алгоритмов генерации ФОС по математике 2024

Курсовая работа

Направление 010501 Фундаментальные математика и механика

Зав.кафедрой _____ д.физ.-мат.н., проф. Е.М. Семёнов

Обучающийся _____ А.С. Суматохина

Руководитель _____ д.физ.-мат.н., проф. Е.М. Семёнов

Воронеж 2024

Содержание

Введение

Единый государственный экзамен (ЕГЭ) — централизованно проводимый в Российской Федерации экзамен в средних учебных заведениях — школах, лицеях и гимназиях, форма проведения ГИА(Государственный Итоговая Аттестация) по образовательным программам среднего общего образования. Служит одновременно выпускным экзаменом из школы и вступительным экзаменом в вузы.

*СТАЩИЛА ИЗ ВЕСНЫ

Но за 10 и 11 класс при подготовке к ЕГЭ школьники сталкиваются с дефицитом заданий по определённым категориям. Так в конце 2021 года в список заданий ЕГЭ были добавлены новые задания под номером 11 по теме "графики функции", а в конце 2023 - задание номер 2 по теме "вектора", количество которых, для прорешивания было очень мало. А по теме "Производная и первообразная" банк заданий с невероятной скоростью.

Так как это преимущественно графические задания, решение их занимает менее минуты, а их составление вручную занимает несоразмерно много времени.

ЕГЭ является относительно неизменяемым экзаменом, поэтому все материалы, которые уже были выложены в открытый доступ имеют полные решения, что приводят к списыванию учениками.

*СТАЩИЛА ИЗ ВЕСНЫ

При этом существуют задания с вспомогательным чертежом. Чаще всего для целого ряда заданий используется одна и та же иллюстрация, которая не всегда соответствуют условиям задачи, а иногда отвлекают от решения. Проект «Час ЕГЭ» позволяет решить все эти проблемы.

«Час ЕГЭ» — компьютерный образовательный проект, разрабатываемый при математическом факультете ВГУ в рамках «OpenSource кластера» и предназначенный для помощи учащимся старших классов подготовиться к тестовой части единого государственного экзамена. Задания в «Час ЕГЭ» генерируются случайным образом по специализированным алгоритмам, называемых шаблонами, каждый из которых охватывает множество вариантов соответствующей ему задачи. Для пользователей предназначены четыре оболочки (режима работы): «Случайное задание», «Тесты на печать», «Полный тест» и «Мини-интеграция». «Час ЕГЭ» является полностью открытым (код находится под лицензией GNU GPL 3.0) и бесплатным. В настоящее время в проекте полностью реализованы тесты по математике с кратким ответом (бывшая «часть B»). [4] Планируется с течением времени включить в проект тесты по другим предметам школьной программы.

«Мини-интеграция» — это форма сотрудничества с образовательными интернет-ресурсами, при которой учебно-методический материал на странице ресурса дополняется виджетами тренажера с заданиями, соответствующими теме статьи, для возможности практического применения полученных знаний. В настоящее время достигнуто сотрудничество с двумя образовательными ресурсами: ege-ok.ru и matematikalegko.ru.

1. Глава первая

1.1. Вспомогательные функции

1.1.1. Функции для работы с массивами

`Array.prototype.permuteCyclic = function(repeat)`
Возвращает массив после циклической перестановки.

```
let array = [1,2,3,4,5];

array.permuteCyclic(1);
// [5, 1, 2, 3, 4]

array.permuteCyclic(-2);
// [3, 4, 5, 1, 2]

array.permuteCyclic(0);
// [1 ,2 ,3 ,4 ,5]
```

`Array.prototype.mt_coordinatesOfIntersectionOfTwoSegments = function()`
Возвращает координаты пересечения двух отрезков, задаваемых первыми парами точек из массива.

```
let array = [{x:0,y:5},{x:-4,y:4},{x:1,y:10},{x:-3,y:6}];

array.mt_coordinatesOfIntersectionOfTwoSegments()
//{ x: -5.333333333333333, y: 3.666666666666667, status: false }
//Отрезки не пересекаются, но прямые проходящие через них пересекаются в точке {x,y}

array = [{x:0,y:5},{x:-4,y:4},{x:1,y:1},{x:-3,y:6}];
array.mt_coordinatesOfIntersectionOfTwoSegments()
//{ x: -1.8333333333333333, y: 4.541666666666667, status: true }
//Отрезки пересекаются в точке {x,y}
```

`Array.prototype.shuffleJoin = function(separator)`
Перемешивает и соединяет массив с разделителем `separator`. `separator` по умолчанию пустая строка.

```
let array = ['A', 'B', 'C', 'D',];
array.shuffleJoin();
//ADBC

array.shuffleJoin('; ');
//C; D; B; A
```

`Array.prototype.joinWithConjunction = function(separator)`
Соединяет массив запятыми и соединяет два последних элемента союзом "и".

```
let array = ['A', 'B', 'C', 'D',];

array.joinWithConjunction();
//A, B, C и D
```

1.1.2. Функции для работы с числами

Number.prototype.perfectCubicMultiplier = function()
Возвращает максимальный делитель данного числа, куб которого также является делителем данного числа.

```
let number = 81;

number.perfectCubicMultiplier()
//3

number = 36;
number.perfectCubicMultiplier()
//1

number = -27;
number.perfectCubicMultiplier()
//3
```

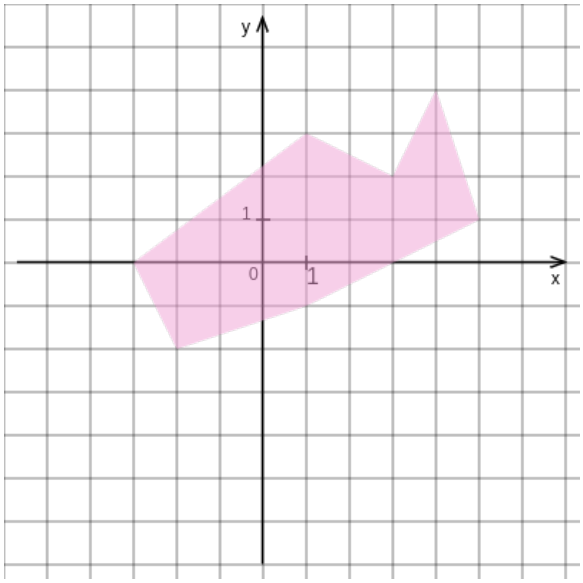
Number.prototype.texcbrt = function(p1, p2)
TeX-представление кубического корня из данного числа.
Если данное число - полный квадрат, то корень из числа.
Если p1, то из-под корня будут вынесены возможные множители.
Если p1, p2 и из-под корня выносится единица, то она будет опущена

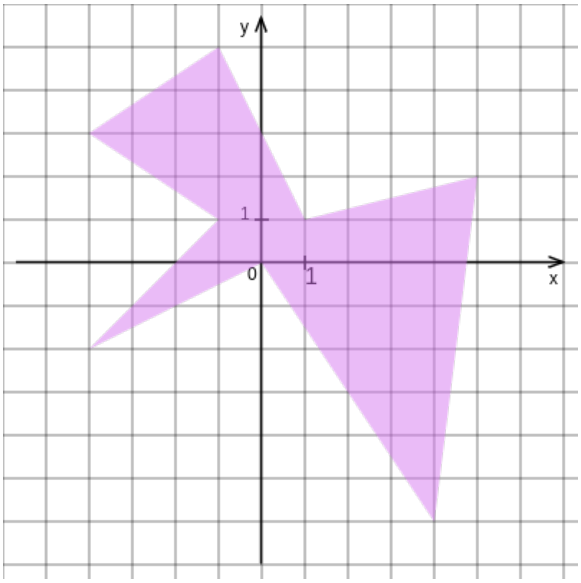
1.1.3. Функции для работы с canvas

CanvasRenderingContext2D.prototype.drawSection = function(vertex, fillStyle)
Заполняет область цветом `fillStyle` по вершинам из массива `vertex`.

```
let paint1 = function(ctx) {
    let h = 400;
    let w = 400;
    ctx.drawCoordinatePlane(w, h, {
        hor: 1,
        ver: 1
    }, {
        x1: '1',
        y1: '1',
        sh1: 16,
    },30);
    ctx.scale(30, -30);
```

```
        ctx.drawSection([[1, 3],[-3, 0],[-2, -2],[1, -1],[5, 1],[4, 4],[3,
    2]]);

        ctx.drawSection([[-2, 0],[-1, 1],[-4, 3],[-1, 5],[1, 1],[5, 2],[4,
    -6],[0, 0],[-4, -2],]);
     };
```

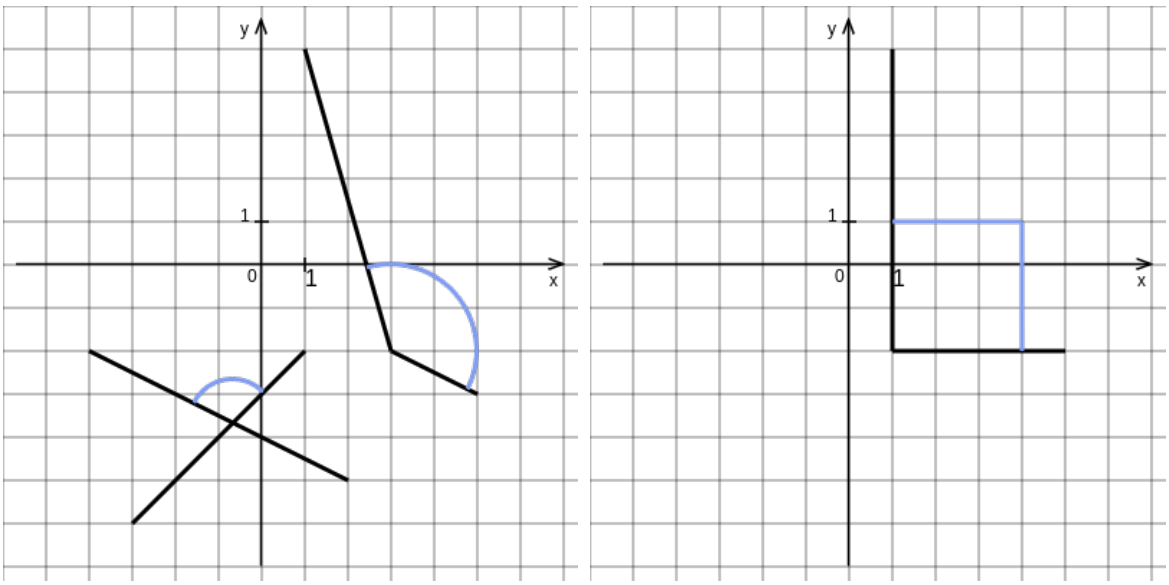




`CanvasRenderingContext2D.prototype.drawLineAtAngle = function(x, y, angle, length)`
Рисует отрезок длины `length` под углом angle в радианах.
`CanvasRenderingContext2D.prototype.strokeInMiddleOfSegment = function(x1, y1, x2, y2, length, quantity)`
Ставит штрихи длины `length` на середине отрезка перпендикулярно ему.
`CanvasRenderingContext2D.prototype.markSegmentWithLetter = function(x, y, angle, letter, length, maxLength)`
Вспомогательная функция для отрисовки текста около некоторого отрезка.
`CanvasRenderingContext2D.prototype.signSegmentInMiddle = function(x1, y1, x2, y2, letter, length, maxLength)`
Рисует `letter` на середине отрезка.
`CanvasRenderingContext2D.prototype.arcBetweenSegments = function(coordinates, radius)`
Рисует знак угла между двумя отрезками в месте их пересечения. `coordinates` - массив вида `[x1, y1, x2, y2]`.

```
    let paint1 = function(ctx) {
        let h = 400;
        let w = 400;
        ctx.drawCoordinatePlane(w, h, {
            hor: 1,
            ver: 1
        }, {
            x1: '1',
            y1: '1',
```

```
            sh1: 16,
        }, 30);
        ctx.scale(30, -30);

        ctx.lineWidth = 2 / 30;
        ctx.drawLine(1, 5, 3, -2);
        ctx.drawLine(3, -2, 5, -3);
        ctx.arcBetweenSegments([1, 5, 3, -2, 5, -3], 2);

        ctx.drawLine(2, -5, -4, -2);
        ctx.drawLine(1, -2, -3, -6);
        ctx.arcBetweenSegments([2, -5, -4, -2,  -3, -6,1, -2,], 1);

        ctx.drawLine(1, 5, 1, -2);
    ctx.drawLine(1, -2, 5, -2);
    ctx.strokeStyle = om.secondaryBrandColors.iz();
    ctx.arcBetweenSegments([1, 5, 1, -2, 5, -2], 3);

    };
```



`CanvasRenderingContext2D.prototype.arcBetweenSegmentsCount = function(coordinates, radius, number, step)`

Рисует знак угла между двумя отрезками в месте их пересечения `number` раз с отступом `step`.

`CanvasRenderingContext2D.prototype.drawEllipse = function(x, y, radiusX, radiusY, rotation, startAngle, endAngle, anticlockwise)`

Рисует эллипс.

`CanvasRenderingContext2D.prototype.drawArc = function(x, y, radius, startAngle, endAngle, anticlockwise)`

Рисует дугу.

2. Глава вторая

2.1. Разработка библиотек с помощью Gpt-Chat

На данный момент в языке JavaScript отсутствуют встроенные средства для изображения трёхмерных фигур. И существует только одна подходящая библиотека `Three.js`, которая могла бы выполнить проецирование координат фигуры на плоскость с учётом положения наблюдателя. Возможности библиотеки во много раз превосходят требуемую функциональность для нашего проекта.

Подобные ей библиотеки проводят проецирование на плоскость с поворотом только вокруг осей OX и OZ. Что при отрисовке более сложных чертежей является недостаточным.

С учётом самых современных технологий, встал закономерный вопрос, сможет ли нейросеть ChatGPT 3.5 сгенерировать код проектора. Ниже представлен диалог с нейросетью:

> write a 2D to 3D projector in JS without using libraries. input: point = {x,y,z}, camera= {x,y,z, angleX, angleY, angleZ, scale}

```
// Define the point in 2D
const point = { x: 100, y: 50, z: 0 };

// Define the camera parameters
const camera = { x: 0, y: 0, z: 0, angleX: 0, angleY: 0, angleZ: 0,
 scale: 1 };

// Function to project 2D point to 3D
function project2Dto3D(point, camera) {
    // Apply rotation around X axis
    const rotatedY = point.y * Math.cos(camera.angleX) - point.z *
 Math.sin(camera.angleX);
    const rotatedZ = point.y * Math.sin(camera.angleX) + point.z *
 Math.cos(camera.angleX);

    // Apply rotation around Y axis
    const rotatedX = point.x * Math.cos(camera.angleY) + rotatedZ *
 Math.sin(camera.angleY);
    const rotatedZ2 = -point.x * Math.sin(camera.angleY) + rotatedZ *
 Math.cos(camera.angleY);

    // Apply rotation around Z axis
    const rotatedX2 = rotatedX * Math.cos(camera.angleZ) - rotatedY *
 Math.sin(camera.angleZ);
    const rotatedY2 = rotatedX * Math.sin(camera.angleZ) + rotatedY *
 Math.cos(camera.angleZ);
```

```
    // Apply scaling
    const scaledX = rotatedX2 * camera.scale;
    const scaledY = rotatedY2 * camera.scale;

    // Translate to camera position
    const translatedX = scaledX + camera.x;
    const translatedY = scaledY + camera.y;
    const translatedZ = rotatedZ2 * camera.scale + camera.z;

    return { x: translatedX, y: translatedY, z: translatedZ };
  }

// Project the point from 2D to 3D
const projectedPoint = project2Dto3D(point, camera);

// Output the result
console.log("Projected 3D point:", projectedPoint);
```

shorten function project2Dto3D

```
function project2Dto3D(point, camera) {
    const { x, y, z } = point;
    const { angleX, angleY, angleZ, scale, x: cx, y: cy, z: cz } =
 camera;

    const cosX = Math.cos(angleX);
    const sinX = Math.sin(angleX);
    const cosY = Math.cos(angleY);
    const sinY = Math.sin(angleY);
    const cosZ = Math.cos(angleZ);
    const sinZ = Math.sin(angleZ);

    const rx = x * cosY * cosZ - y * cosY * sinZ + z * sinY;
    const ry = (x * (sinX * sinY * cosZ + cosX * sinZ) + y * (sinX *
 sinY * sinZ - cosX * cosZ) + z * sinX * cosY) * scale;
    const rz = (-x * (cosX * sinY * cosZ - sinX * sinZ) - y * (cosX *
 sinY * sinZ + sinX * cosZ) + z * cosX * cosY) * scale;

    return { x: rx + cx, y: ry + cy, z: rz + cz };
  }
```

add comments in code

```
function project2Dto3D(point, camera) {
    // Extract point coordinates
```

```
    const { x, y, z } = point;
    // Extract camera parameters
    const { angleX, angleY, angleZ, scale, x: cx, y: cy, z: cz } =
camera;

    // Precompute trigonometric values for rotation angles
    const cosX = Math.cos(angleX);
    const sinX = Math.sin(angleX);
    const cosY = Math.cos(angleY);
    const sinY = Math.sin(angleY);
    const cosZ = Math.cos(angleZ);
    const sinZ = Math.sin(angleZ);

    // Apply rotation around X axis
    const rx = x * cosY * cosZ - y * cosY * sinZ + z * sinY;
    // Apply rotation around Y axis and scale
    const ry = (x * (sinX * sinY * cosZ + cosX * sinZ) + y * (sinX *
sinY * sinZ - cosX * cosZ) + z * sinX * cosY) * scale;
    // Apply rotation around Z axis and scale
    const rz = (-x * (cosX * sinY * cosZ - sinX * sinZ) - y * (cosX *
sinY * sinZ + sinX * cosZ) + z * cosX * cosY) * scale;

    // Translate to camera position
    const translatedX = rx + cx;
    const translatedY = ry + cy;
    const translatedZ = rz + cz;

    return { x: translatedX, y: translatedY, z: translatedZ };
  }
```

За несколько шагов удалось получить корректный, оптимизированный код.

2.2. Применение ООП для разработки шаблонов

Стоит отметить, что задач по теме "Стереометрия" огромное множество. Поэтому одной из первостепенных задач было сократить код шаблонов и исключить вычислительные ошибки. Для этого были разработаны классы многогранников, которые содержат в себе длины рёбер, объем, площади оснований, а так же тернарную матрицу связности и канонические координаты вершин.

Матрица может содержать значения: 1, 0, либо специальное значение, указывающий на отображении ребра пунктиром.

Пример канонической матрицы связей:

```
[   [1],
    [0, 1],
    [1, 0, 1],
    [0, 0, 0, 1],
    [1, 0, 0, 0, 1],
    [0, 1, 0, 0, 0, 1],
```

```
    [0, 0, 1, 0, 1, 0, 1],
];
```

Листинг 1: Каноническая матрица связей для параллелепипеда

Каноническим положением будем называть такое расположение многогранника, когда его высота, проходящая через центр масс его основания, совпадает с осью аппликат и начало координат делится пополам.

При таком расположении, начало координат можно расположить в центре иллюстрация. Тогда чертёж не будет смещён ни в одну из сторон.

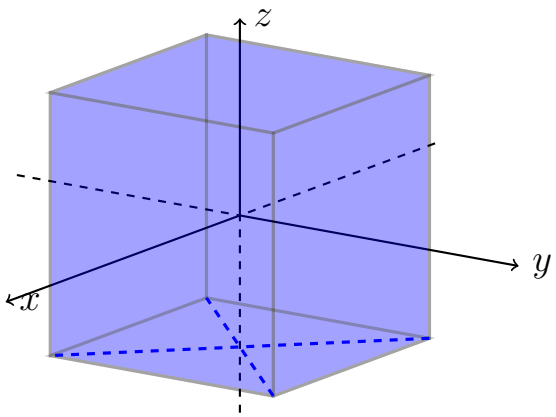


Рис. 1: Каноническое положение для параллелепипеда

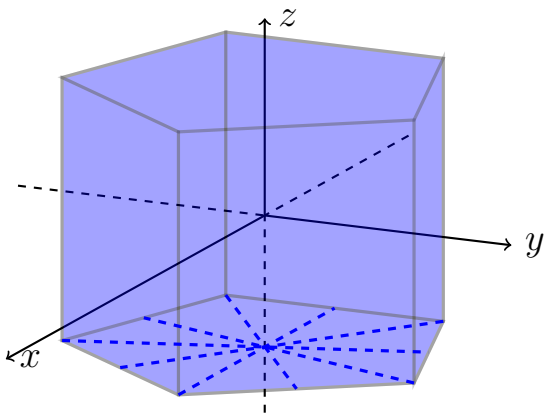


Рис. 2: Каноническое положение для правильной пятиугольной призмы

2.3. Вспомогательные функции

2.3.1. Функции для работы с координатами

`function verticesInGivenRange(vertex, startX, finishX, startY, finishY)`
Возвращает `true`, если двухмерная координата точки `vertex` вида `{x,y}` находится в некоторой области, иначе `false`.

`function autoScale(vertex3D, camera, vertex2D, startX, finishX, startY, finishY, step, maxScale)`
Увеличивает свойство объекта `camera.scale`, пока все двухмерные координаты `vertex2D` вида `{x,y}` находится в некоторой области. `step` по умолчанию 0.1.

`function distanceFromPointToSegment(point, segmentStart, segmentEnd)`
Возвращает длину перпендикуляра между двухмерной точкой `point` вида `{x,y}` до отрезка с концами в `segmentStart` и `segmentEnd`.

2.3.2. Функции для работы с canvas

`CanvasRenderingContext2D.prototype.drawFigure = function(vertex, matrixC`
Соединяет линиями точки массива `vertex` с элементами `{x,y}` в соответствии с матрицей связей `matrixConnections`, которая является массивом,который может содержать в себе 0, 1 и массив step, указывающий на отрисовку пунктиром.

Пример матрицы связей:

```
let matrixConnections = [
    [1],
    [strok, strok],
    [0, 0, strok],
    [1, 0, 0, 1],
    [0, 1, 0, 1, 1]
 ];
```

`CanvasRenderingContext2D.prototype.drawFigureVer2 = function()` vertex, matrixConnections Соединяет линиями точки массива `vertex` с элементами `{x,y}` в соответствии с матрицей связей `matrixConnections`, которая является объектом с числовыми полями (номерами вершин), которые содержат в себе массив с номерами вершин для связи с ними.

Пример матрицы связей:

```
let matrixConnections = {
  0: [1, [3, stroke], 5],
  2: [1, [3, stroke], 7],
  4: [[3, stroke], 5, 7],
  9: [1, 8, 10],
  11: [8, 10, 12],
  13: [5, 8, 12],
  15: [7, 10, 12],
};
```

2.4. Этапы разработки шаблоны с вспомогательным чертежом

Для примера возьмём задание 27074

```
(function () {
  retryWhileError(function () {
    NAinfo.requireApiVersion(0, 2);

    let paint1 = function (ctx) {
    };

    NAtask.setTask({
      text: 'Объем параллелепипеда ABCDA_1B_1C_1D_1 равен 9. Найдите объем
   треугольной пирамиды ABCA_1.',
      answers: 0,
      author: ['Суматохина Александра']
    });
    NAtask.modifiers.addCanvasIllustration({
      width: 400,
      height: 400,
      paint: paint1,
    });
  }, 100000);
})();
```

1. Создадим объект класса `Parallelepiped`.

```
(function () {
  retryWhileError(function () {
    NAinfo.requireApiVersion(0, 2);
```

```
    let par = new Parallelepiped({
      depth: sl(10, 50),
      height: sl(10, 50),
      width: sl(10, 50),
    });

    let paint1 = function (ctx) {
    };

    NAtask.setTask({
      text: 'Объем параллелепипеда ABCDA_1B_1C_1D_1 равен 9. Найдите
   объем треугольной пирамиды ABCA_1.',
      answers: 0,
      author: ['Суматохина Александра']
    });
    NAtask.modifiers.addCanvasIllustration({
      width: 400,
      height: 400,
      paint: paint1,
    });
  }, 100000);
})();
```

2. Определим переменную `camera`, которая будет отвечать за положение наблюдателя. И спроецируем канонические координаты параллелепипеда на двухмерную плоскость при помощи функции `project3DTo2D`. И отмасштабируем полученные координаты так, чтобы они занимали максимально заполняли спрайт, функцией `autoScale`.

```
(function () {
    retryWhileError(function () {
        NAinfo.requireApiVersion(0, 2);

        let par = new Parallelepiped({
            depth: sl(10, 50),
            height: sl(10, 50),
            width: sl(10, 50),
        });

        let camera = {
            x: 0,
            y: 0,
            z: 0,
            scale: 5,

            rotationX: -Math.PI / 2 + Math.PI / 14,
            rotationY: 0,
            rotationZ: 2 * Math.PI / 3,
        };

        let point2DPar = par.verticesOfFigure.map((coord3D) =>
```

```
    project3DTo2D(coord3D, camera));

        autoScale(par.verticesOfFigure, camera, point2DPar, {
            startX: -180,
            finishX: 160,
            startY: -160,
            finishY: 160,
            maxScale: 50,
        });

        point2DPar = par.verticesOfFigure.map((coord3D) =>
    project3DTo2D(coord3D, camera));

        let paint1 = function (ctx) {
        };

        NAtask.setTask({
            text: 'Объем параллелепипеда ABCDA_1B_1C_1D_1 равен 9.
    Найдите объем треугольной пирамиды ABCA_1.',
            answers: 0,
            author: ['Суматохина Александра']
        });
        NAtask.modifiers.addCanvasIllustration({
            width: 400,
            height: 400,
            paint: paint1,
        });
     }, 100000);
})();
```

3. Перемещаемся в середину спрайта. Отрисовываем фигуру функцией `drawFigure`, отдав в неё матрицу связей для параллелепипеда.

```
(function () {
    retryWhileError(function () {
        NAinfo.requireApiVersion(0, 2);

        let par = new Parallelepiped({
            depth: sl(10, 50),
            height: sl(10, 50),
            width: sl(10, 50),
        });

        let camera = {
            x: 0,
            y: 0,
            z: 0,
            scale: 5,

            rotationX: -Math.PI / 2 + Math.PI / 14,
            rotationY: 0,
            rotationZ: 2 * Math.PI / 3,
```

```
        };

        let point2DPar = par.verticesOfFigure.map((coord3D) =>
    project3DTo2D(coord3D, camera));

        autoScale(par.verticesOfFigure, camera, point2DPar, {
            startX: -180,
            finishX: 160,
            startY: -160,
            finishY: 160,
            maxScale: 50,
        });

        point2DPar = par.verticesOfFigure.map((coord3D) =>
    project3DTo2D(coord3D, camera));

        let paint1 = function (ctx) {
            let h = 400;
            let w = 400;
            ctx.translate(h / 2, w / 2);
            ctx.lineWidth = 2;
            ctx.strokeStyle = om.secondaryBrandColors;
            let strok = [5, 4];
            ctx.drawFigure(point2DPar, [
                [strok],
                [0, 1],
                [strok, 0, 1],
                [0, 0, 0, 1],
                [strok, 0, 0, 0, 1],
                [0, 1, 0, 0, 0, 1],
                [0, 0, 1, 0, 1, 0, 1],
            ]);
        };

        NAtask.setTask({
            text: 'Объем параллелепипеда ABCDA_1B_1C_1D_1 равен 9.
    Найдите объем треугольной пирамиды ABCA_1.',
            answers: 0,
            author: ['Суматохина Александра']
        });
        NAtask.modifiers.addCanvasIllustration({
            width: 400,
            height: 400,
            paint: paint1,
        });
    }, 100000);
})();
```

4. Далее вырезаем из условия значения и заменяем их данными из класса. Впишем ответ. Обособляем имена фигур в $$. Добавляем буквы на вершины параллелепипеда. Добавим модификаторы `NAtask.modifiers.assertSaneDecimals`

(исключает нецелый ответ) и `NAtask.modifiers.variativeABC(letter)` (заменяет все буквы в задании на случайные).

```
(function() {
  retryWhileError(function() {
    NAinfo.requireApiVersion(0, 2);

    let par = new Parallelepiped({
      depth: sl(10, 50),
      height: sl(10, 50),
      width: sl(10, 50),
    });

    let pyr = new Pyramid({
      height: par.height,
      baseArea: 0.5 * par.baseArea,
    });

    let camera = {
      x: 0,
      y: 0,
      z: 0,
      scale: 5,

      rotationX: -Math.PI / 2 + Math.PI / 14,
      rotationY: 0,
      rotationZ: 2 * Math.PI / 3,
    };

    let point2DPar = par.verticesOfFigure.map((coord3D) =>
   project3DTo2D(coord3D, camera));

    autoScale(par.verticesOfFigure, camera, point2DPar, {
      startX: -180,
      finishX: 160,
      startY: -160,
      finishY: 160,
      maxScale: 50,
    });

    point2DPar = par.verticesOfFigure.map((coord3D) =>
   project3DTo2D(coord3D, camera));

    let letter = ['A', 'B', 'C', 'D', '□D', '□A', '□B', '□C',];

    let paint1 = function(ctx) {
      let h = 400;
      let w = 400;
      ctx.translate(h / 2, w / 2);
      ctx.lineWidth = 2;
      ctx.strokeStyle = om.secondaryBrandColors;
      let strok = [5, 4];
```

```
      ctx.drawFigure(point2DPar, [
        [strok],
        [0, 1],
        [strok, 0, 1],
        [0, 0, 0, 1],
        [strok, 0, 0, 0, 1],
        [0, 1, 0, 0, 0, 1],
        [0, 0, 1, 0, 1, 0, 1],
      ]);

      ctx.font = "25px liberation_sans";
            point2DPar.forEach((elem, i) => ctx.fillText(letter[i],
   elem.x, elem.y + ((i < point2DPar.length / 2) ? 15 : -10)));
    };

    NAtask.setTask({
      text: 'Объем параллелепипеда $ABCDA_1B_1C_1D_1$ равен
   $'+par.volume+'$. Найдите объем треугольной пирамиды $ABCA_1$.',
      answers: par.volume/6,
      author: ['Суматохина Александра']
    });

    NAtask.modifiers.assertSaneDecimals();
    NAtask.modifiers.variativeABC(letter);

    NAtask.modifiers.addCanvasIllustration({
      width: 400,
      height: 400,
      paint: paint1,
    });
  }, 100000);
})();
```

Заключение

В данной работе были приведены архитектура проекта «Час ЕГЭ», его библиотеки и примеры генерируемых задач. Обоснована релевантность проекта по сравнению с другими открытыми ресурсами.

Список литературы

[1] Момот Е. А., Арахов Н. Д. Разработка и внедрение ПО для сбора статистики результатов подготовки к ЕГЭ по математике профильного уровня //Актуальные проблемы прикладной математики, информатики и механики. – 2021. – С. 1-2.

[2] Открытый банк задач ЕГЭ по Математике.Профильный уровень. – URL: https://prof.mathege.ru/

[3] Пошаговая инструкция по созданию элементарных шаблонов. – URL: https://math.vsu.ru/chas-ege/doc/shabl-b1-po-shagam.html

[4] Федеральный институт педагогических измерений. – URL: https://fipi.ru/ege/otkrytyy-bank-zadaniy-ege

[5] Единый государственный экзамен. – URL: https://ru.wikipedia.org/wiki/Единый_государственный_экзамен

Приложение

Шаблоны по теме Планиметрия

Шаблоны по теме Стереометрия

```
(function() {
  retryWhileError(function() {
    lx_declareClarifiedPhrase('сторона', 'основания');
    NAinfo.requireApiVersion(0, 2);

    let pyr = new RegularPyramid({
      height: sl(20, 50),
      baseSide: sl(20, 40),
      numberSide: 4
    });

    pyr.verticesOfFigure.push({
      x: 0,
      y: 0,
      z: pyr.verticesOfFigure[0].z
    });

    let question = [
      [sklonlxkand('боковое ребро'), pyr.sideEdge],
      [sklonlxkand('объём'), pyr.volume],
    ].shuffle();
    question.unshift([sklonlxkand('сторона основания'), pyr.baseSide]);

    let camera = {
      x: 0,
      y: 0,
      z: 0,
      scale: 5,

      rotationX: -Math.PI / 2 + Math.PI / 14,
      rotationY: 0,
      rotationZ: [1, 2].iz() * Math.PI / 3,
    };

    let point2DPyr = pyr.verticesOfFigure.map((coord3D) =>
   project3DTo2D(coord3D, camera));

    autoScale(pyr.verticesOfFigure, camera, point2DPyr, {
      startX: -180,
      finishX: 160,
      startY: -160,
      finishY: 160,
      maxScale: 50,
    });
```

```
    point2DPyr = pyr.verticesOfFigure.map((coord3D) =>
  project3DTo2D(coord3D, camera));

    let letters = ['A', 'B', 'C', 'D', 'S', 'O'];

    let strok = [5, 4];


    let paint1 = function(ctx) {
      let h = 400;
      let w = 400;
      ctx.translate(h / 2, w / 2);
      ctx.lineWidth = 2;
      ctx.strokeStyle = om.secondaryBrandColors;
      ctx.drawFigure(point2DPyr, [
        [1],
        [strok, strok],
        [1, strok, strok],
        [1, 1, strok, 1, 0, strok],
      ]);

      ctx.font = "30px liberation_sans";
      point2DPyr.forEach((elem, i) => ctx.fillText(letters[i], elem.x,
  elem.y + ((i != point2DPyr.length - 2) ? 15 : -
        10)));
    };

    NAtask.setTask({
      text: 'В правильной четырёхугольной пирамиде $SABCD$ с ' + 'основанием
  $ABCD$ ' +
        [question[0][0].ie + ' рав' + ['ен', 'на',
  'но'][question[0][0].rod] + ' $' + question[0][1].pow(2).texsqrt(1) +
  '$',
        question[1][0].ie + ' рав' + ['ен', 'на', 'но'][question[1][0].rod]
  + ' $' + question[1][1].pow(2).texsqrt(1) +'$'
        ].shuffleJoin(', ') +
        '. Найдите ' + question[2][0].ve + ' пирамиды.',
      answers: question[2][1],
      author: ['Суматохина Александра'],
    });
    NAtask.modifiers.variativeABC(letters);
    NAtask.modifiers.multiplyAnswerBySqrt(13);
    NAtask.modifiers.allDecimalsToStandard(true);
    NAtask.modifiers.assertSaneDecimals();
    NAtask.modifiers.addCanvasIllustration({
      width: 400,
      height: 400,
      paint: paint1,
    });
  },10);
})();
//https://ege314.ru/8-stereometriya-ege/reshenie-3011/
```

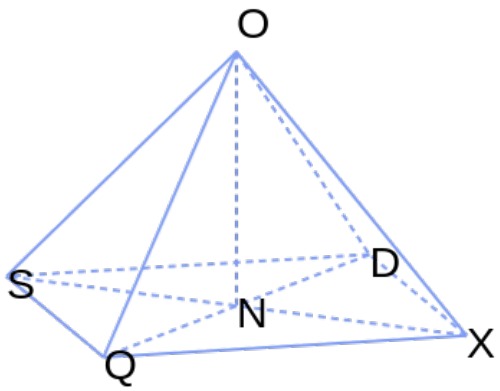
```
//3011
```

Листинг 2: 3011.js

Примеры генерируемых задач 3011.js

В правильной четырёхугольной пирамиде $OQSDX$ с основанием $QSDX$ боковое ребро равно $\sqrt{1489{,}5}$, сторона основания равна 39. Найдите объём пирамиды.

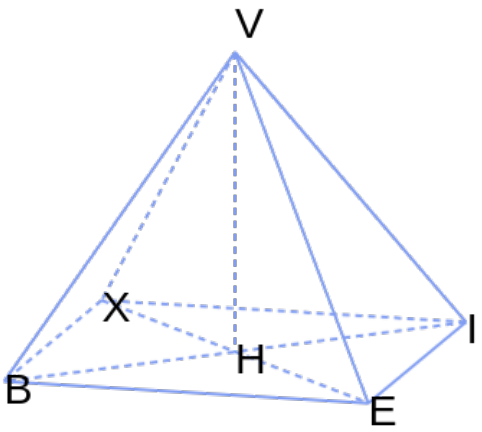
Ответ: 13689



Приложение. 1

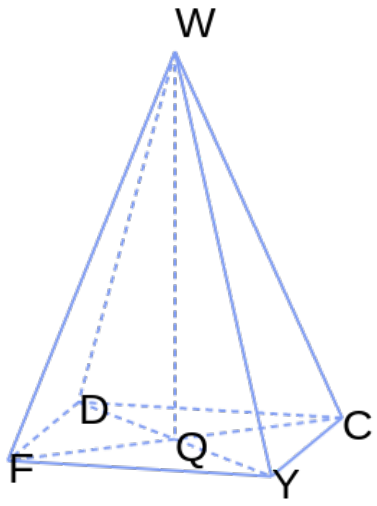
В правильной четырёхугольной пирамиде $VEBXI$ с основанием $EBXI$ боковое ребро равно $\sqrt{848{,}5}$, сторона основания равна 27. Найдите объём пирамиды.

Ответ: 5346



Приложение. 2

В правильной четырёхугольной пирамиде $WYFDC$ с основанием $YFDC$ боковое ребро равно $\sqrt{1513}$, сторона основания равна 24. Найдите объём пирамиды.



Ответ: 6720

Приложение. 3

```
(function() {
  retryWhileError(function() {

    let pyr1 = new RegularPyramid({
      height: sl(30, 70),
      baseSide: sl(20, 50),
      numberSide: 4
    });

    let pyr2 = new Pyramid({
      height: 0.5 * pyr1.height,
      baseArea: 0.5 * pyr1.baseArea,
    });

    pyr1.verticesOfFigure =
   coordinatesMiddleOfSegment3D(pyr1.verticesOfFigure[0],
   pyr1.verticesOfFigure[4]);

    let camera = {
      x: 0,
      y: 0,
      z: 0,
      scale: 5,

      rotationX: -Math.PI / 2 + Math.PI / 14,
      rotationY: 0,
      rotationZ: [1, 2].iz() * Math.PI / 3,
    };

    let point2DPyr = pyr1.verticesOfFigure.map((coord3D) =>
   project3DTo2D(coord3D, camera));

    autoScale(pyr1.verticesOfFigure, camera, point2DPyr, {
```

```
    startX: -180,
    finishX: 160,
    startY: -160,
    finishY: 160,
    maxScale: 50,
  });

  point2DPyr = pyr1.verticesOfFigure.map((coord3D) =>
 project3DTo2D(coord3D, camera));

  let letters = ['A', 'B', 'C', 'D', 'S', 'E'];
  let strok = [5, 4];

  let paint1 = function(ctx) {
    let h = 400;
    let w = 400;
    ctx.translate(h / 2, w / 2);
    ctx.lineWidth = 2;
    ctx.strokeStyle = om.secondaryBrandColors;
    ctx.drawFigure(point2DPyr, [
      [1],
      [0, strok],
      [1, strok, strok],
      [1, 1, strok, 1, ],
      [0, 1, 0, 1, 0]
    ]);

    ctx.font = "30px liberation_sans";
    point2DPyr.forEach((elem, i) => ctx.fillText(letters[i], elem.x,
 elem.y + ((i <= point2DPyr.length - 3) ? 15 : -
      10)));

  };

  NAtask.setTask({
    text: 'Объём правильной четырёхугольной пирамиды $SABCD$ равен $' +
 pyr1.volume.pow(2).texsqrt(1) + '$. ' +
      'Точка $E$ - середина ребра $SA$. Найдите объём треугольной пирамиды
 $EABD$.',
    answers: pyr2.volume,
    author: ['Суматохина Александра'],
  });
  NAtask.modifiers.variativeABC(letters);
  NAtask.modifiers.multiplyAnswerBySqrt(13);
  NAtask.modifiers.allDecimalsToStandard(true);
  NAtask.modifiers.assertSaneDecimals();
  NAtask.modifiers.addCanvasIllustration({
    width: 400,
    height: 400,
    paint: paint1,
  });
});
```

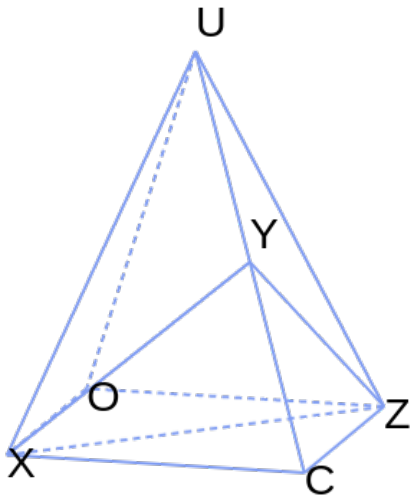
```
})();
//27114 75015 75063 519535 75017 75019 75021 75023 75025 75027 75029 75031
   75033 75035 75037 75039 75041 75043 75045 75047 75049 75051 75053 75055
   75057 75059 75061
```

Листинг 3: 27114.js

Примеры генерируемых задач 3011.js

Объём правильной четырёхугольной пирамиды $UCXOZ$ равен 31164. Точка Y – середина ребра UC. Найдите объём треугольной пирамиды $YCXZ$.

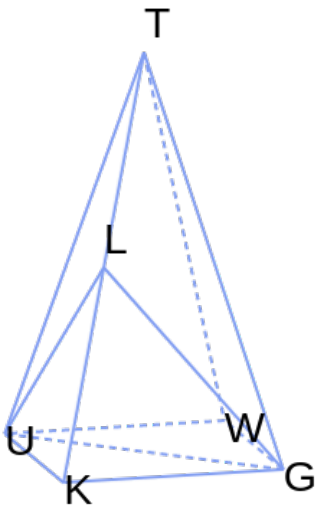
Ответ: 7791



Приложение. 4

Объём правильной четырёхугольной пирамиды $TKUWG$ равен 4800. Точка L – середина ребра TK. Найдите объём треугольной пирамиды $LKUG$.

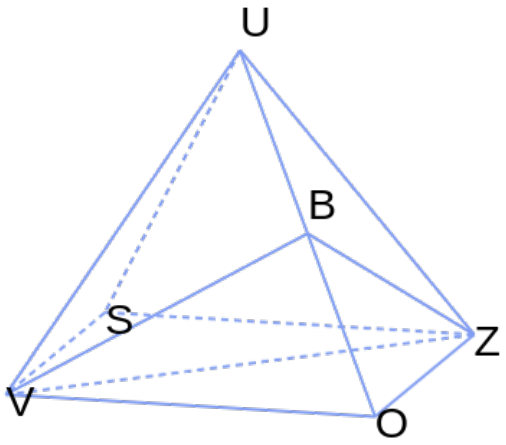
Ответ: 1200



Приложение. 5

Объём правильной четырёхугольной пирамиды $UOVSZ$ равен 25650. Точка B – середина ребра UO. Найдите объём треугольной пирамиды $BOVZ$.

Ответ: 6412,5



Приложение. 6

```
(function() {
  retryWhileError(function() {

    let pyr1 = new RegularPyramid({
      height: sl(10, 30)*(3).sqrt(),
      baseSide: sl(20, 50),
      numberSide: 3
    });

    let pyr2 = new RegularPyramid({
      height: pyr1.height,
      baseSide: 0.5 * pyr1.baseSide,
      numberSide: 3
    });


   pyr1.verticesOfFigure.push(coordinatesMiddleOfSegment3D(pyr1.verticesOfFigure
   pyr1.verticesOfFigure[1]));

   pyr1.verticesOfFigure.push(coordinatesMiddleOfSegment3D(pyr1.verticesOfFigure
   pyr1.verticesOfFigure[2]));

    let strok = [5, 4];

    let camera = {
      x: 0,
      y: 0,
      z: 0,
      scale: 5,

      rotationX: -Math.PI / 2 + Math.PI / 14,
      rotationY: 0,
      rotationZ: sl(1,2)* Math.PI / 8,
```

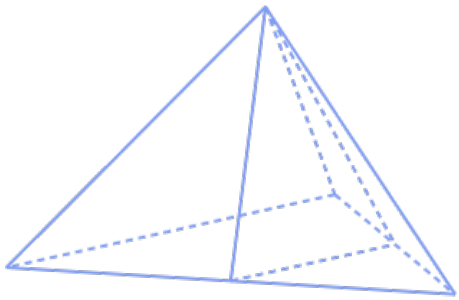
```
    };

    let point2DPyr = pyr1.verticesOfFigure.map((coord3D) =>
   project3DTo2D(coord3D, camera));

    autoScale(pyr1.verticesOfFigure, camera, point2DPyr, {
      startX: -180,
      finishX: 160,
      startY: -160,
      finishY: 160,
      maxScale: 50,
    });

    point2DPyr = pyr1.verticesOfFigure.map((coord3D) =>
   project3DTo2D(coord3D, camera));

    let paint1 = function(ctx) {
      let h = 400;
      let w = 400;
      ctx.translate(h / 2, w / 2);
      ctx.lineWidth = 2;
      ctx.strokeStyle = om.secondaryBrandColors;
      ctx.drawFigure(point2DPyr, [
        [1],
        [strok, strok],
        [1, 1, strok, 0, 1, strok],
        [0, 0, 0, 0, 0, strok]
      ]);
    };

    NAtask.setTask({
      text: ' Объём треугольной пирамиды равен $' + pyr1.volume + '$. ' +
        'Через вершину пирамиды и среднюю линию её основания проведена плоскость
   см(. рисунок). ' +
        'Найдите объём отсечённой треугольной пирамиды.',
      answers: pyr2.volume,
      author: ['Суматохина Александра'],
    });
    NAtask.modifiers.multiplyAnswerBySqrt(13);
    NAtask.modifiers.allDecimalsToStandard(true);
    NAtask.modifiers.assertSaneDecimals();
    NAtask.modifiers.addCanvasIllustration({
      width: 400,
      height: 400,
      paint: paint1,
    });
  }, 1000);
})();
//27115 75065 75109 75113 514460 75067 75069 75071 75073 75075 75077 75079
   75081 75083 75085 75087 75089 75091 75093 75095 75097 75099 75101 75103
   75105 75107 75111
```

Листинг 4: 3011.js

Примеры генерируемых задач 27115.js

Объём треугольной пирамиды равен 2560. Через вершину пирамиды и среднюю линию её основания проведена плоскость (см. рисунок). Найдите объём отсечённой треугольной пирамиды.

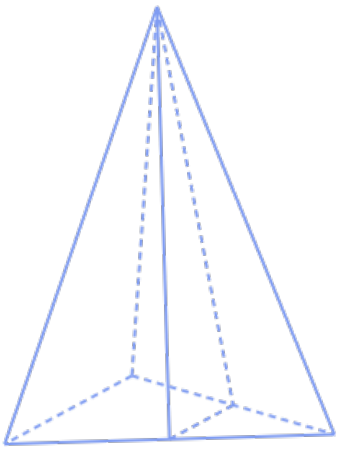
Ответ: 640



Приложение. 7

Объём треугольной пирамиды равен 4950. Через вершину пирамиды и среднюю линию её основания проведена плоскость (см. рисунок). Найдите объём отсечённой треугольной пирамиды.

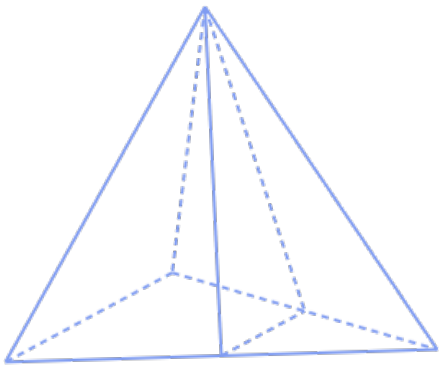
Ответ: 1237,5



Приложение. 8

Объём треугольной пирамиды равен 12096. Через вершину пирамиды и среднюю линию её основания проведена плоскость (см. рисунок). Найдите объём отсечённой треугольной пирамиды.

Ответ: 3024



Приложение. 9

```
(function() {
  retryWhileError(function() {
      NAinfo.requireApiVersion(0, 2);

      let stroke = [4, 5];

      let matrixConnections = {
        0: [1, [3, stroke], 5],
        2: [1, [3, stroke], 7],
        4: [[3, stroke], 5, 7],
        9: [1, 8, 10],
        11: [8, 10, 12],
        13: [5, 8, 12],
        15: [7, 10, 12],
      };

      let par1 = new Parallelepiped({
        depth: sl(10, 20),
        height: sl(10, 20),
        width: sl(10, 20),
      });

      let par2 = new Parallelepiped({
        depth: slKrome(par1.depth, 5, par1.depth - 3),
        height: slKrome(par1.height, 5, par1.height - 3),
        width: slKrome(par1.width, 5, par1.width - 5),
      });

      let vertex3D =
   par1.verticesOfFigure.concat(par2.verticesOfFigure.map((elem) =>
   shiftCoordinate3D(elem, {
        x: -0.5 * (par1.width - par2.width),
        y: -0.5 * (par1.depth - par2.depth),
```

```
      z: -0.5 * (par1.height - par2.height),
    })));

    let camera = {
      x: 0,
      y: 0,
      z: 0,
      scale: 1,

      rotationX: -Math.PI / 2 + Math.PI / 14,
      rotationY: 0,
      rotationZ: Math.PI / sl(10, 14),
    };

    let point2D = vertex3D.map((coord3D) => project3DTo2D(coord3D,
camera));

    autoScale(vertex3D, camera, point2D, {
      startX: -180,
      finishX: 160,
      startY: -160,
      finishY: 160,
      maxScale: 100,
    });

    point2D = vertex3D.map((coord3D) => project3DTo2D(coord3D, camera));
    genAssert((point2D[3].y - point2D[11].y).abs() > 20);
    genAssert((point2D[3].y - point2D[8].y).abs() > 20);
    genAssert((point2D[13].x - point2D[14].x).abs() > 20);

    let rand = sl1();

    let paint1 = function(ctx) {
      let h = 400;
      let w = 400;
      ctx.translate(w / 2, h / 2);
      ctx.lineWidth = 2;
      ctx.strokeStyle = om.secondaryBrandColors;
      ctx.drawFigureVer2(point2D, matrixConnections);

      ctx.font = "20px liberation_sans";
      ctx.signSegmentInMiddle(point2D[4].x, point2D[4].y, point2D[7].x,
point2D[7].y, par1.height, -10, 20);
      ctx.signSegmentInMiddle(point2D[10].x, point2D[10].y,
point2D[15].x, point2D[15].y, par2.height, -20, 20);
      ctx.signSegmentInMiddle(point2D[12].x, point2D[12].y,
point2D[15].x, point2D[15].y, par2.width, 20, 20);
      ctx.signSegmentInMiddle(point2D[0].x, point2D[0].y, point2D[1].x,
point2D[1].y, par1.width, 18, 20);
      ctx.signSegmentInMiddle(point2D[1].x, point2D[1].y, point2D[2].x,
point2D[2].y, par1.depth, 18, 20);
      ctx.signSegmentInMiddle(point2D[9].x, point2D[9].y, point2D[10].x,
```

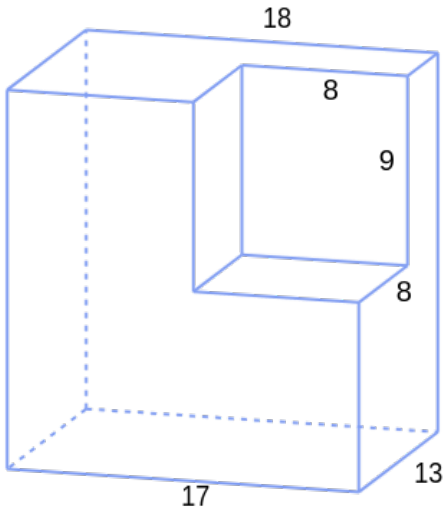
```
  point2D[10].y, par2.depth, 15, 20);
      };
      NAtask.setTask({
        text: 'Найдите ' + ['площадь поверхности', 'объём'][rand] +
          ' многогранника, изображённого на рисунке все( двугранные углы -
  прямые).',
        answers: [par1.surfaceArea, par1.volume - par2.volume][rand],
      });
      NAtask.modifiers.addCanvasIllustration({
        width: 400,
        height: 400,
        paint: paint1,
      });
    },
    1000);

})();
//27193 25671 25673 25675 25677 25679
```

Листинг 5: 12.js

Примеры генерируемых задач 12.js

Найдите объём многогранника, изображённого на рисунке (все двугранные углы – прямые).

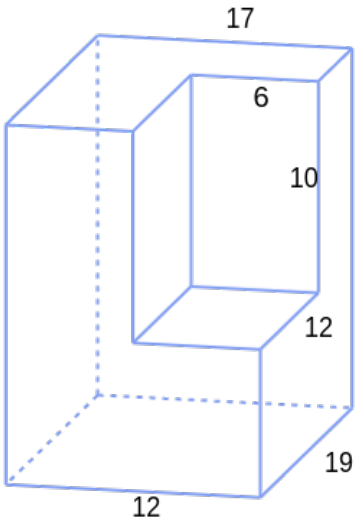
Ответ: 3402



Приложение. 10

Найдите объём многогранника, изображённого на рисунке (все двугранные углы – прямые).

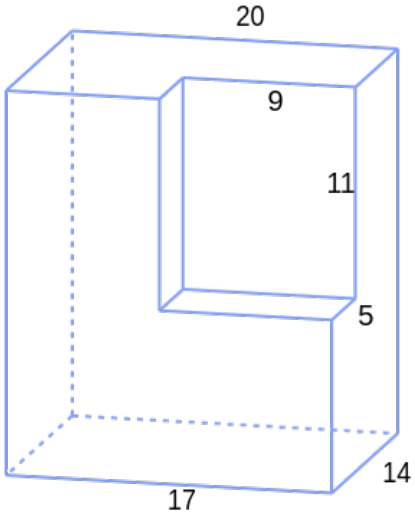
Ответ: 3156



Приложение. 11

Найдите объём многогранника, изображённого на рисунке (все двугранные углы – прямые).

Ответ: 4265



Приложение. 12

```
(function() {
  retryWhileError(function() {
      NAinfo.requireApiVersion(0, 2);

      let stroke = [4, 5];

      let matrixConnections = {
        0: [1, [3, stroke], 5],
        2: [1, [3, stroke], 7],
        4: [
          [3, stroke],
          [5, stroke],
          [11, stroke]
        ],
        6: [1, 7, 9],
```

```
      8: [5, [11, stroke], 13],
      10: [7, 9, 15],
      12: [
        [11, stroke], 13, 15
      ],
      14: [9, 13, 15],
    };

    let par1 = new Parallelepiped({
      depth: sl(10, 20),
      height: sl(5, 20),
      width: sl(10, 20),
    });

    let par2 = new Parallelepiped({
      depth: par1.depth,
      height: sl(5, 20),
      width: slKrome(par1.width, 5, par1.width - 5),
    });

    let vertex3D =
par1.verticesOfFigure.concat(par2.verticesOfFigure.map((elem) =>
shiftCoordinate3D(elem, {
      x: 0,
      y: 0,
      z: -0.5 * (par1.height + par2.height),
    })));

    vertex3D = vertex3D.map((elem) => shiftCoordinate3D(elem, {
      x: 0,
      y: 0,
      z: 0.5 * par2.height,
    }));

    let camera = {
      x: 0,
      y: 0,
      z: 0,
      scale: 1,

      rotationX: -Math.PI / 2 + Math.PI / 14,
      rotationY: 0,
      rotationZ: Math.PI / sl(10, 14),
    };

    let point2D = vertex3D.map((coord3D) => project3DTo2D(coord3D,
camera));

    autoScale(vertex3D, camera, point2D, {
      startX: -180,
      finishX: 160,
      startY: -160,
```

```
      finishY: 160,
      maxScale: 100,
    });

    point2D = vertex3D.map((coord3D) => project3DTo2D(coord3D, camera));
    genAssert((point2D[4].x - point2D[8].x).abs() > 20);
    genAssert((point2D[4].y - point2D[13].y).abs() > 50);
    genAssert((point2D[12].x - point2D[14].x).abs() > 40);

    let rand = sl1();

    let paint1 = function(ctx) {
      let h = 400;
      let w = 400;
      ctx.translate(w / 2, h / 2);
      ctx.lineWidth = 2;
      ctx.strokeStyle = om.secondaryBrandColors;
      ctx.drawFigureVer2(point2D, matrixConnections);

      if (point2D[4].x > point2D[8].x) {
        let point = [point2D[4], point2D[5], point2D[8],
point2D[13]].mt_coordinatesOfIntersectionOfTwoSegments();
        ctx.drawLine(point2D[5].x, point2D[5].y, point.x, point.y);
      } else {
        ctx.drawLine(point2D[4].x, point2D[4].y, point2D[5].x,
point2D[5].y);
        let point = [point2D[4], point2D[11], point2D[8],
point2D[13]].mt_coordinatesOfIntersectionOfTwoSegments();
        ctx.drawLine(point2D[4].x, point2D[4].y, point.x, point.y);
      }

      ctx.font = "20px liberation_sans";
      ctx.signSegmentInMiddle(point2D[2].x, point2D[2].y, point2D[7].x,
point2D[7].y, par1.height, 10, 20);
      ctx.signSegmentInMiddle(point2D[10].x, point2D[10].y,
point2D[15].x, point2D[15].y, par2.height, 10, 20);
      ctx.signSegmentInMiddle(point2D[12].x, point2D[12].y,
point2D[15].x, point2D[15].y, par2.width, -10, 20);
      ctx.signSegmentInMiddle(point2D[0].x, point2D[0].y, point2D[1].x,
point2D[1].y, par1.width, 18, 20);
      ctx.signSegmentInMiddle(point2D[1].x, point2D[1].y, point2D[2].x,
point2D[2].y, par1.depth, 18, 20);
    };
    NAtask.setTask({
      text: 'Найдите ' + ['площадь поверхности', 'объём'][rand] +
        ' многогранника, изображённого на рисунке все( двугранные углы -
прямые).',
      answers: [par1.surfaceArea + par2.surfaceArea - 2 * par2.baseArea,
par1.volume + par2.volume][rand],
    });
    NAtask.modifiers.addCanvasIllustration({
      width: 400,
```

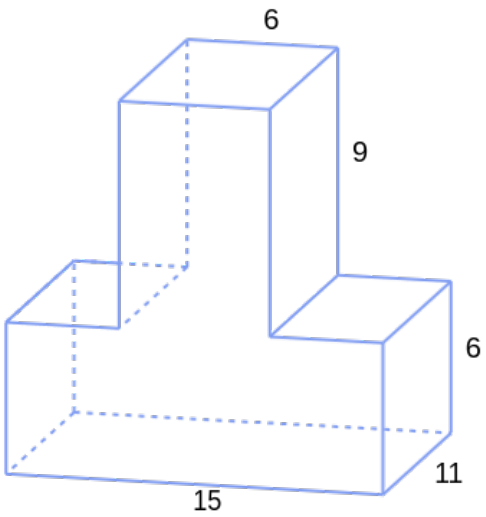
```
        height: 400,
        paint: paint1,
      });
    },
    1000);

})();
//27193 25671 25673 25675 25677 25679
```

Листинг 6: 29193.js

Примеры генерируемых задач 27193.js

Найдите площадь поверхности многогранника, изображённого на рисунке (все двугранные углы – прямые).

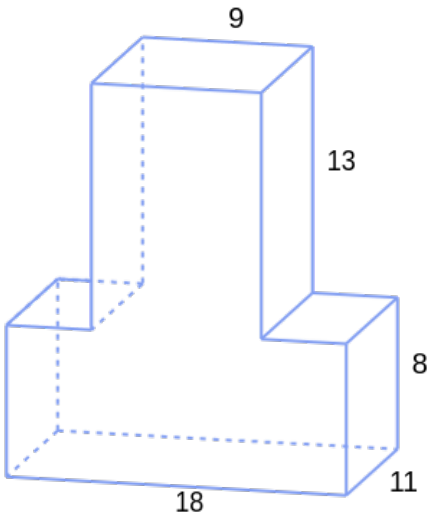
Ответ: 948



Приложение. 13

Найдите объём многогранника, изображённого на рисунке (все двугранные углы – прямые).

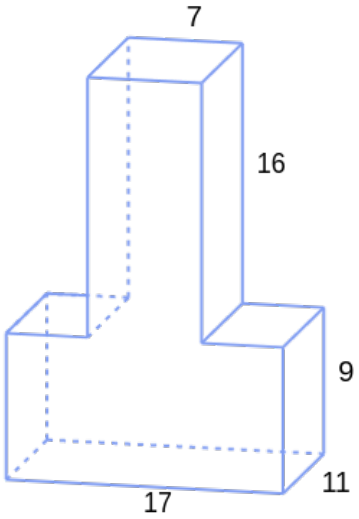
Ответ: 2871



Приложение. 14

Найдите площадь поверхности многогранника, изображённого на рисунке (все двугранные углы – прямые).

Ответ: 1454



Приложение. 15

```
(function() {
  retryWhileError(function() {
      NAinfo.requireApiVersion(0, 2);

      let stroke = [4, 2];

      let matrixConnections = {
        0: [1, [3, stroke], 5],
        2: [1, [3, stroke], 7],
        4: [
          [3, stroke], 5, [11, stroke]
        ],
        6: [1, 7, 9],
        8: [5, [11, stroke], 13],
        10: [
          [7, stroke],
          [9, stroke],
          [15, stroke]
        ],
        12: [
          [11, stroke], 13, 15
        ],
        14: [9, 13, 15],
      };

      let par1 = new Parallelepiped({
        depth: sl(10, 20),
        height: sl(10, 20),
        width: sl(10, 20),
      });

      let par2 = new Parallelepiped({
        depth: par1.depth,
```

```
      height: sl(5, par1.height - 4),
      width: slKrome(par1.width, 5, par1.width - 5),
    });

    let deltaWidth = par1.width - par2.width;
    let diagonal = (0.25 * deltaWidth.pow(2) +
par2.height.pow(2)).sqrt();

    let vertex3D =
par1.verticesOfFigure.concat(par2.verticesOfFigure.map((elem) =>
shiftCoordinate3D(elem, {
      x: 0,
      y: 0,
      z: -0.5 * (par1.height - par2.height),
    })));

    let camera = {
      x: 0,
      y: 0,
      z: 0,
      scale: 1,

      rotationX: -Math.PI / 2 + Math.PI / 14,
      rotationY: 0,
      rotationZ: Math.PI / sl(12, 14),
    };

    let point2D = vertex3D.map((coord3D) => project3DTo2D(coord3D,
camera));

    autoScale(vertex3D, camera, point2D, {
      startX: -180,
      finishX: 160,
      startY: -160,
      finishY: 160,
      maxScale: 100,
    });

    point2D = vertex3D.map((coord3D) => project3DTo2D(coord3D, camera));
    genAssert((point2D[4].x - point2D[8].x).abs() > 20);
    genAssert((point2D[4].y - point2D[13].y).abs() > 50);
    genAssert((point2D[12].x - point2D[14].x).abs() > 40);
    genAssert([point2D[0], point2D[3], point2D[8]].mt_is3ug(), 'Точки
лежат на одной прямой');

    let rand = sl1();

    let paint1 = function(ctx) {
      let h = 400;
      let w = 400;
      ctx.translate(w / 2, h / 2);
      ctx.lineWidth = 2;
```

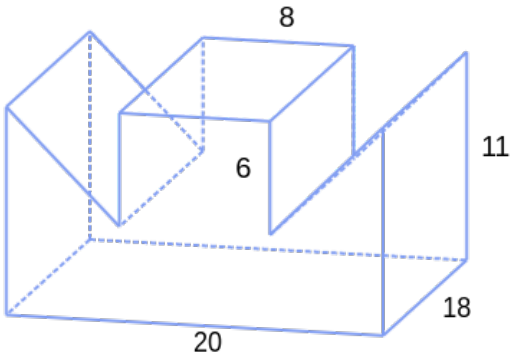
```
        ctx.strokeStyle = om.secondaryBrandColors;
        ctx.drawFigureVer2(point2D, matrixConnections);

        let point = [];
        ctx.drawLine(point2D[4].x, point2D[4].y, point.x, point.y);
        if (point2D[6].x < point2D[15].x)
          point = [point2D[10], point2D[15], point2D[6],
    point2D[7]].mt_coordinatesOfIntersectionOfTwoSegments();
        else
          point = [point2D[10], point2D[15], point2D[6],
    point2D[9]].mt_coordinatesOfIntersectionOfTwoSegments();
        ctx.drawLine(point2D[15].x, point2D[15].y, point.x, point.y);

        point = [point2D[12], point2D[13], point2D[4],
    point2D[11]].mt_coordinatesOfIntersectionOfTwoSegments();
        ctx.drawLine(point2D[4].x, point2D[4].y, point.x, point.y);

        ctx.font = "20px liberation_sans";
        ctx.signSegmentInMiddle(point2D[2].x, point2D[2].y, point2D[7].x,
    point2D[7].y, par1.height, 10, 20);
        ctx.signSegmentInMiddle(point2D[9].x, point2D[9].y, point2D[14].x,
    point2D[14].y, par2.height, -24, 20);
        ctx.signSegmentInMiddle(point2D[12].x, point2D[12].y,
    point2D[15].x, point2D[15].y, par2.width, -10, 20);
        ctx.signSegmentInMiddle(point2D[0].x, point2D[0].y, point2D[1].x,
    point2D[1].y, par1.width, 18, 20);
        ctx.signSegmentInMiddle(point2D[1].x, point2D[1].y, point2D[2].x,
    point2D[2].y, par1.depth, 18, 20);
      };

      NAtask.setTask({
        text: 'Найдите ' + ['площадь поверхности', 'объём'][rand] + '
    многогранника, изображённого на рисунке.',
        answers: [par1.surfaceArea - deltaWidth * (par2.height +
    par2.depth) + 2 * par2.height * par2.depth + 2 *
          diagonal * par2.depth,
          par1.volume - par2.height * (par1.width - par2.width) *
    par2.width
        ][rand],
        author: ['Суматохина Александра']
      });
      NAtask.modifiers.multiplyAnswerBySqrt(13);
      NAtask.modifiers.addCanvasIllustration({
        width: 400,
        height: 400,
        paint: paint1,
      });
    },
    1000);

})();
//144526144
```

Листинг 7: 144526144.js

Примеры генерируемых задач 144526144.js

Найдите площадь поверхности многогранника, изображённого на рисунке.

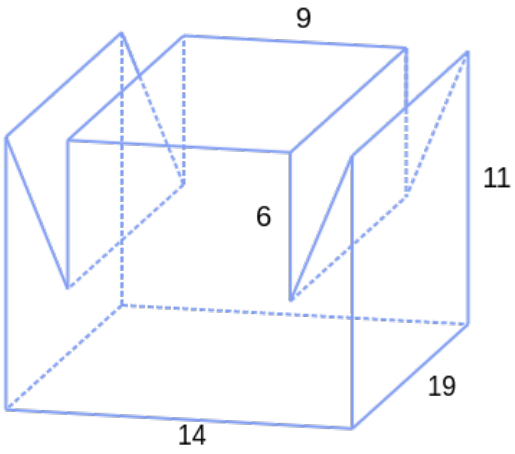
Ответ: 3384



Приложение. 16

Найдите площадь поверхности многогранника, изображённого на рисунке.

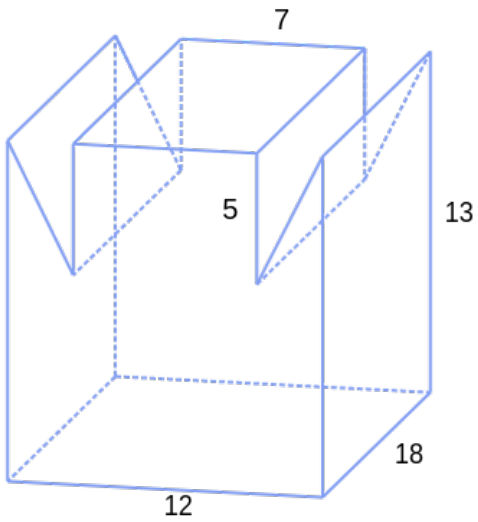
Ответ: 2656



Приложение. 17

Найдите площадь поверхности многогранника, изображённого на рисунке.

Ответ: 2633



Приложение. 18