

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 ЕГЭ по геометрии в 2024 году

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	3
1 Планиметрия	5
1.1 Вспомогательные функции	5
1.1.1 Функции для работы с массивами	5
1.1.2 Функции для работы с числами	6
1.1.3 Функции для работы с canvas	6
1.1.4 Элементы декларативного программирования	9
1.2 Этапы разработки шаблона со вспомогательным чертежом по теме «Планиметрия»	10
2 Стереометрия	14
2.1 Разработка библиотек с помощью Gpt-Chat	14
2.2 Применение объектно-ориентированного программирования для раз- работки шаблонов	17
2.3 Вспомогательные функции	18
2.3.1 Функции для работы с координатами	18
2.3.2 Функции для работы с canvas	18
2.4 Этапы разработки шаблоны с вспомогательным чертежом по теме «Стереометрия»	19
Заключение	25
Приложение	27

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.

Но за время обучения в 10 и 11 классе 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 функций», а в конце 2023 — задание №2 по теме «Вектора», количество которых для прорешивания было очень мало. А по теме «Производная и первообразная» банк заданий расходуется при подготовке 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 приводит к списыванию учениками.

При этом существуют задания с вспомогательным чертежом. Чаще всего для целого ряда заданий используется одна и та же иллюстрация, которая не всегда соответствует условиям задачи, а иногда отвлекает 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 при подготовке 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 В»). [3] Планируется с течением времени включить в проект тесты по другим предметам школьной программы.

Первая глава этой работы посвящена обзору вспомогательных функций, которые ускоряют написание шаблонов по теме «Планиметрия» и введению в проект элементов декларативного программирования. Также приведён алгоритм написания шаблона с чертежом.

Вторая глава представляет решение проблемы отрисовки фигур в трёхмерном пространстве на языке программирования JavaScript; рассказывает о применении объектно-ориентированного программирования для упрощения написания шаблонов с чертежом; затрагивает вопрос об написании программного кода при помощи нейросетей; приводит обзор вспомогательных функций и алгоритм написания

шаблона по теме «Стереометрия».

1. Планиметрия

В этой главе мы приводим вспомогательные функции и алгоритм написания шаблона по планиметрии

1.1. Вспомогательные функции

1.1.1. Функции для работы с массивами

`Array.prototype.permuteCyclic = function(repeat)`

Возвращает массив после циклической перестановки. В листинге 8 в строке 38 функция используется для перестановки букв в названии параллелограмма, а в строке 40 — в названии трапеции.

```
1  let array = [1, 2, 3, 4, 5];
2
3  array.permuteCyclic(1);
4  // [5, 1, 2, 3, 4]
5
6  array.permuteCyclic(-2);
7  // [3, 4, 5, 1, 2]
8
9  array.permuteCyclic(0);
10 // [1, 2, 3, 4, 5]
11
```

`Array.prototype.mt_coordinatesOfIntersectionOfTwoSegments = function()`

Возвращает координаты пересечения двух отрезков, задаваемых первыми парами точек из массива. Является вспомогательной для функции `arcBetweenSegments`.

```
1  let array = [{x:0,y:5},{x:-4,y:4},{x:1,y:10},{x:-3,y:6}];
2
3  array.mt_coordinatesOfIntersectionOfTwoSegments()
4  //{ x: -5.333333333333333, y: 3.6666666666666667, status: false }
5  //Если status - false, отрезки не пересекаются, но прямые проходящие через
   них пересекаются в точке {x,y}
6
7  array = [{x:0,y:5},{x:-4,y:4},{x:1,y:1},{x:-3,y:6}];
8  array.mt_coordinatesOfIntersectionOfTwoSegments()
9  //{ x: -1.8333333333333333, y: 4.5416666666666667, status: true }
10 //Если 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. Функция используется в 5 в строке 74 для отображения условий задачи в случайном порядке.

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

```

3 //ADBC
4
5 array.shuffleJoin('; ');
6 //C; D; B; A

```

`Array.prototype.joinWithConjunction = function(separator)`

Соединяет массив запятыми и соединяет два последних элемента союзом «и».

```

1 let array = ['A', 'B', 'C', 'D',];
2
3 array.joinWithConjunction();
4 //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.

```

1 let number = 81;
2
3 number.perfectCubicMultiplier()
4 //3
5
6 number = 36;
7 number.perfectCubicMultiplier()
8 //1
9
10 number = -27;
11 number.perfectCubicMultiplier()
12 //3

```

`Number.prototype.texcbirt = function(p1, p2)`

TeX-представление кубического корня из данного числа.

Если данное число - полный куб, 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

Радиусы трёх шаров равны $\sqrt[3]{3}$, $\sqrt[3]{6}$, $\sqrt[3]{18}$. Найдите радиус шара, объем которого равен сумме их объемов.

1.1.3. Функции для работы с canvas

`CanvasRenderingContext2D.prototype.drawSection = function(vertex, fillStyle)`

Заполняет область цветом fillStyle по вершинам из массива vertex.

```

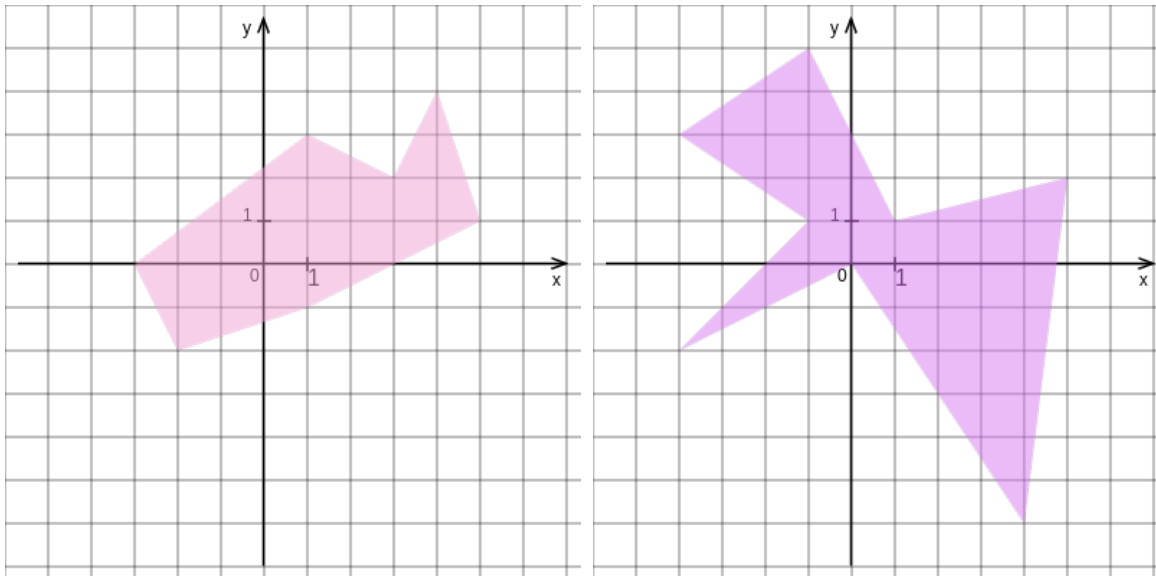
1 let paint1 = function(ctx) {
2     let h = 400;

```

```

3      let w = 400;
4      ctx.drawCoordinatePlane(w, h, {
5          hor: 1,
6          ver: 1
7      }, {
8          x1: '1',
9          y1: '1',
10         sh1: 16,
11     }, 30);
12     ctx.scale(30, -30);
13     ctx.drawSection([[1, 3], [-3, 0], [-2, -2], [1, -1], [5, 1], [4, 4], [3,
14     2]]);
15     ctx.drawSection([[[-2, 0], [-1, 1], [-4, 3], [-1, 5], [1, 1], [5, 2], [4,
16     -6], [0, 0], [-4, -2], ]]);
17     };

```



`CanvasRenderingContext2D.prototype.drawLineAtAngle = function(x, y, angle, length)`

Рисует отрезок длины `length` под углом `angle` (в радианах). Пример использования можно найти в листинге 6 в строках 19 и 25 (применяется для отрисовки биссектрисы).

`CanvasRenderingContext2D.prototype.strokeInMiddleOfSegment = function(x1, y1, x2, y2, length, quantity)`

Ставит штрихи длины `length` на середине отрезка перпендикулярно ему. Функция используется в листинге 6 в строках 21-22 для обозначения равных по длине сторон треугольника.

`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)`

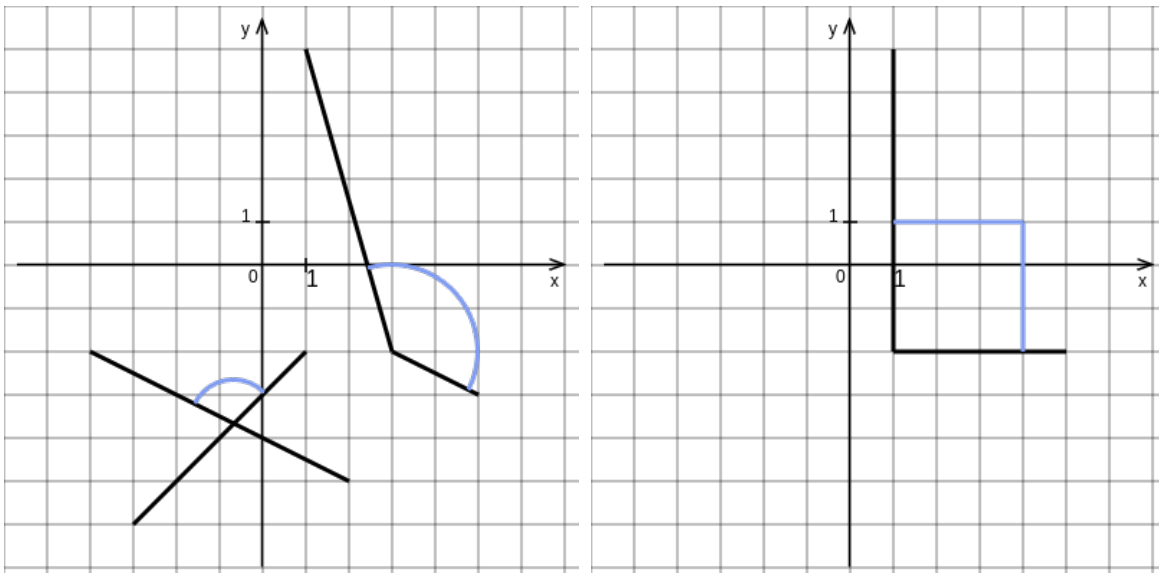
Рисует строку `letter` на середине отрезка. В листинге 4 в строках 94 - 98 функция используется для отображения длин рёбер многогранника.

`CanvasRenderingContext2D.prototype.arcBetweenSegments`

`= function(coordinates, radius)`

Рисует знак угла между двумя отрезками в месте их пересечения. `coordinates` - массив вида `[x1, y1, x2, y2]`.

```
1  let paint1 = function(ctx) {
2    let h = 400;
3    let w = 400;
4    ctx.drawCoordinatePlane(w, h, {
5      hor: 1,
6      ver: 1
7    }, {
8      x1: '1',
9      y1: '1',
10     sh1: 16,
11   }, 30);
12   ctx.scale(30, -30);
13
14   ctx.lineWidth = 2 / 30;
15   ctx.drawLine(1, 5, 3, -2);
16   ctx.drawLine(3, -2, 5, -3);
17   ctx.arcBetweenSegments([1, 5, 3, -2, 5, -3], 2);
18
19   ctx.drawLine(2, -5, -4, -2);
20   ctx.drawLine(1, -2, -3, -6);
21   ctx.arcBetweenSegments([2, -5, -4, -2, -3, -6, 1, -2,], 1);
22
23   ctx.drawLine(1, 5, 1, -2);
24   ctx.drawLine(1, -2, 5, -2);
25   ctx.strokeStyle = om.secondaryBrandColors.iz();
26   ctx.arcBetweenSegments([1, 5, 1, -2, 5, -2], 3);
27
28   };
```




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

Рисует знак угла между двумя отрезками в месте их пересечения `number` раз с отступом `step`. В листинге 7 в строках 27 - 28 используется для обозначения двух равных углов.

```
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.

1.1.4. Элементы декларативного программирования

Определение. Декларативное программирование — парадигма программирования, в которой задается спецификация решения задачи, то есть описывается конечный результат, а не способ его достижения.

Во время разработки шаблонов по теме «Графики функции» требовалось много раз переопределять коэффициенты функций через циклы `while` или `do...while`, пока они не начнут соответствовать заданным условиям (видимость графика, сливание его с осями, видимость целых точек). Это часто приводило к бесконечной работе шаблона, при этом сложно было определить, какое условие не выполняется.

Для этого было разработано окружение `retryWhileUndefined` для шаблонов, которое бы перезапускало их не более `maxIterations` раз, если одно из условий не удовлетворено.

```
function retryWhileUndefined(theFunction, maxIterations)
```

Но всё равно было тяжело определить, почему шаблон перезапускается. Для этого было разработано более совершенное окружение `retryWhileError`, которое не только могло бы ограничивать количество перезапусков, но и фиксировать, какие проверки не были пройдены и выводить их на экран (ошибки видны только для разработчика при отладке).

```
function retryWhileError(theFunction, maxIterations, maxCollectedErrors)
```

Для окружения были написаны функции-утверждения, которые имеют структуру : условие не выполнено - записать ошибку - перезапустить шаблон. Если максимальное количество повторений достигнуто, то вывести накопившиеся ошибки и количество их появлений.

```
function genAssert(condition, message)
```

Если условие `condition` ложно, то шаблон перезапускается.

```
function genAssertNonempty(array, message)
```

Если массив `array` пуст, то шаблон перезапускается.

```
function genAssertZ1000(number, message)
```

Если число `number` имеет более 3 знаков после запятой, то шаблон перезапускается.

```
function genAssertIrreducible(numerator, denominator, message)
```

Если дробь `numerator/denominator` сократима, то шаблон перезапускается.

```
function genAssertSaneDecomposition(number, maxFactor, message)
```

Если `number` число не раскладывается на простые множители, не более одного из которых превосходит `maxFactor`, то шаблон перезапускается.

1.2. Этапы разработки шаблона со вспомогательным чертежом по теме «Планиметрия»

Для примера возьмём задание №19416 [2].

Задача №19376. В треугольнике ABC известно, что $AC = BC$, $AB = 16$, AH — высота, $BH = 4$. Найдите косинус угла BAC .

Заготовка шаблона имеет вид.

```
1 (function () {
2   retryWhileError(function () {
3     NAinfo.requireApiVersion(0, 2);
4
5     let paint1 = function (ctx) {
6       };
7
8     NATask.setTask({
9       text: 'В треугольнике ABC AC=BC, AB=15, AHвысота-, BH=5. Найдите косинус
ABC',
10      answers: 0,
11      author: ['Суматохина Александра']
12    });
13    NATask.modifiers.addCanvasIllustration({
14      width: 400,
15      height: 400,
16      paint: paint1,
17    });
18  }, 1000);
19 })();
```

1. Начнём с отрисовки чертежа для задания. Отметим стороны треугольника так, чтобы он лежал в центре холста, а до краёв оставалось 10-20px. При отрисовке используем функцию `drawLine`. Добавим высоту

```
1 (function () {
2   retryWhileError(function () {
3     NAinfo.requireApiVersion(0, 2);
4
5     let paint1 = function (ctx) {
```

```

6
7     ctx.lineWidth = 2;
8     ctx.strokeStyle = om.secondaryBrandColors;
9
10    //стороны
11    ctx.drawLine(10, 370, 390, 370);
12    ctx.drawLine(10, 370, 180, 50);
13    ctx.drawLine(180, 50, 390, 370);
14    //высота
15    ctx.drawLine(280, 200, 10, 370);
16 };
17
18 NATask.setTask({
19     text: 'В треугольнике ABC AC=BC, AB=15, AHвысота-, BHНайдите=5.
косинус ABC.',
20     answers: 0,
21     author: ['Суматохина Александра']
22 });
23 NATask.modifiers.addCanvasIllustration({
24     width: 400,
25     height: 400,
26     paint: paint1,
27 });
28 }, 1000);
29 })();

```

2. Добавим на рисунок штрихи, указывающие на равенство сторон и обозначение прямого угла при помощи функций `strokeInMiddleOfSegment` и `arcBetweenSegments` соответственно. И подпишем вершины и точку перпендикуляра. Добавим модификатор `NATask.modifiers.variativeABC(vertices)`, который заменяет все буквы в задании на случайные.

```

1 (function () {
2     retryWhileError(function () {
3         NAinfo.requireApiVersion(0, 2);
4
5         let vertices = ['A', 'B', 'C', 'H'];
6
7         let paint1 = function (ctx) {
8
9             ctx.lineWidth = 2;
10            ctx.strokeStyle = om.secondaryBrandColors;
11
12            //стороны
13            ctx.drawLine(10, 370, 390, 370);
14            ctx.drawLine(10, 370, 180, 50);
15            ctx.drawLine(180, 50, 390, 370);
16            //высота
17            ctx.drawLine(280, 200, 10, 370);
18
19            ctx.strokeInMiddleOfSegment(180, 50, 10, 370, 10);

```

```

20     ctx.strokeInMiddleOfSegment(180, 50, 390, 370, 10);
21
22     ctx.arcBetweenSegments([180, 50, 280, 200, 280, 200, 10, 370],
23                             25);
24
25     ctx.font = "23px liberation_sans";
26     ctx.fillText(vertices[0], 10 - 5, 370 + 25);
27     ctx.fillText(vertices[1], 180, 50 - 10);
28     ctx.fillText(vertices[2], 390 - 10, 370 + 25);
29     ctx.fillText(vertices[3], ver2.x + 10, ver2.y);
30
31     NATask.setTask({
32         text: 'В треугольнике ABC AC=BC, AB=15, AHвысота-, BHНайдите=5.
33             косинус ABC.',
34         answers: 0,
35         author: ['Суматохина Александра']
36     });
37     NATask.modifiers.variativeABC(vertices);
38     NATask.modifiers.addCanvasIllustration({
39         width: 400,
40         height: 400,
41         paint: paint1,
42     }, 1000);
43 })();

```

3. Теперь добавим ответ в задание. Проверим при помощи `genAssertZ1000`, что ответ имеет не более трёх знаков после запятой (если иначе шаблон запускается заново). Поместим все буквы и числа в `$...$`. Все условия из задачи преобразуем в массив и соединим случайным образом с помощью функции `shuffleJoin`.

```

1  (function () {
2      retryWhileError(function () {
3          NAinfo.requireApiVersion(0, 2);
4
5          let a = sl(2, 89);
6          let b = slKrome(a, 1, a - 1);
7          genAssertZ1000(b / a, 'Нецелый ответ');
8
9          let vertices = ['A', 'B', 'C', 'H'];
10
11         let paint1 = function (ctx) {
12
13             ctx.lineWidth = 2;
14             ctx.strokeStyle = om.secondaryBrandColors;
15
16             //стороны
17             ctx.drawLine(10, 370, 390, 370);
18             ctx.drawLine(10, 370, 180, 50);

```

```

19     ctx.drawLine(180, 50, 390, 370);
20     //высота
21     ctx.drawLine(280, 200, 10, 370);
22
23     ctx.strokeInMiddleOfSegment(180, 50, 10, 370, 10);
24     ctx.strokeInMiddleOfSegment(180, 50, 390, 370, 10);
25
26     ctx.arcBetweenSegments([180, 50, 280, 200, 280, 200, 10, 370],
27                             25);
28
29     ctx.font = "23px liberation_sans";
30     ctx.fillText(vertices[0], 10 - 5, 370 + 25);
31     ctx.fillText(vertices[1], 180, 50 - 10);
32     ctx.fillText(vertices[2], 390 - 10, 370 + 25);
33     ctx.fillText(vertices[3], 280 + 10, 200);
34
35     NATask.setTask({
36         text: 'В треугольнике $ABC$ ' + ['$AC=BC$', '$AB=' + a + '$',
37         '$AHвысота-$', '$BH=' + b + '$'].shuffleJoin(', ') + '. Найдите
38         косинус $BAC$.',
39         answers: b / a,
40         author: ['Суматохина Александра']
41     });
42     NATask.modifiers.variativeABC(vertices);
43     NATask.modifiers.addCanvasIllustration({
44         width: 400,
45         height: 400,
46         paint: paint1,
47     });
48 }, 1000);
49 })();

```

2. Стереометрия

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. Но при этом для создания любого объекта необходима не только камера, но и сцена с рендерингом, что значительно замедляет работу проекта.

```
1 import * as THREE from 'three';
2
3 // Создать сцену, камеру и рендер
4 const scene = new THREE.Scene();
5 const camera = new THREE.PerspectiveCamera(75, window.innerWidth /
   window.innerHeight, 0.1, 1000);
6 camera.position.z = 2;
7 const renderer = new THREE.WebGLRenderer();
8 renderer.setSize(window.innerWidth, window.innerHeight);
9
10 // Создать геометрию куба
11 const geometry = new THREE.BoxGeometry(1, 1, 1);
12
13 // Создать материал куба
14 const material = new THREE.MeshBasicMaterial({ color: 0x00ff00 });
15
16 // Создать куб
17 const cube = new THREE.Mesh(geometry, material);
18
19 // Добавить куб в сцену
20 scene.add(cube);
21
22 // Отрендерить сцену
23 renderer.render(scene, camera);
```

Листинг 1: Код, необходимый для отрисовки куба

Другие 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 прогресса 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 объектно-ориентированного программирования для разработки шаблонов

Банк заданий содержит большое количество разнообразных задач по теме «Стереометрия». 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, указывающие на отображение ребра пунктиром.

Пример канонической матрицы связей:

```

1  [  [1],
2     [0, 1],
3     [1, 0, 1],
4     [0, 0, 0, 1],
5     [1, 0, 0, 0, 1],
6     [0, 1, 0, 0, 0, 1],
7     [0, 0, 1, 0, 1, 0, 1],
8 ];

```

Листинг 2: Каноническая матрица связей для параллелепипеда

Мы можем опускать конец матрицы, если он состоит только из нулей, и главную диагональ матрицы (на ней всегда стоят нули).

Определение. Каноническим положением будем называть такое расположение многогранника, когда его высота, проходящая через центр масс его основания, совпадает с осью аппликата и началом координат делится пополам (Рис. 1-Рис. 2).

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

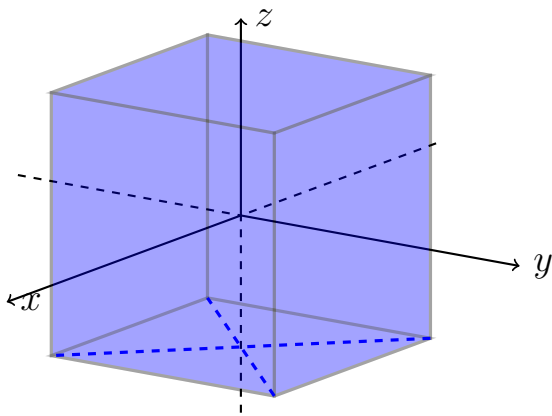


Рис. 1: Каноническое положение для параллелепипеда

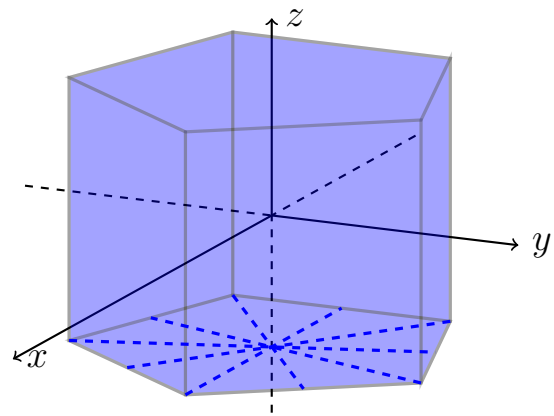


Рис. 2: Каноническое положение для правильной пятиугольной призмы

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 прямоугольной области, иначе **false**.

function autoScale(vertex3D, camera, vertex2D, startX, finishX, startY, finishY, step, maxScale)

Увеличивает свойство объекта **camera.scale** до тех пор, пока все двухмерные координаты **vertex2D** вида {x,y} находится в некоторой прямоугольной области. **step** по умолчанию 0.1.

function distanceFromPointToSegment(point, segmentStart, segmentEnd)

Возвращает длину перпендикуляра между двухмерной точкой **point** вида {x,y} до прямой, проходящей через точки **segmentStart** и **segmentEnd**.

2.3.2. Функции для работы с canvas

CanvasRenderingContext2D.prototype.drawFigure = **function**(vertex, matrixConnections)

Соединяет линиями точки массива **vertex** с элементами {x,y} в соответствии с матрицей связей **matrixConnections**, которая является массивом, содержащим в себе 0, 1 или 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)`

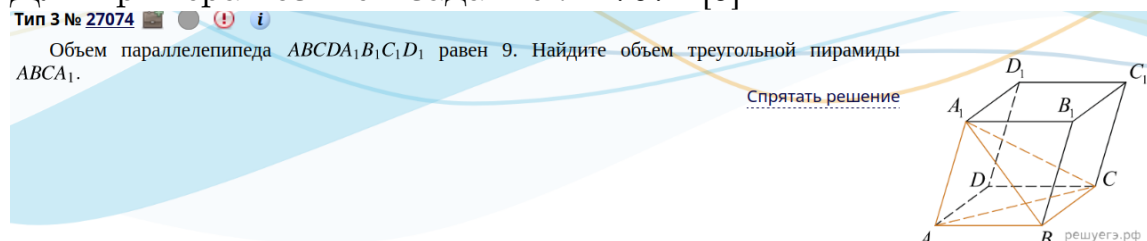
Соединяет линиями точки массива `vertex` с элементами `{x,y}` в соответствии с матрицей связей `matrixConnections`. Эта матрица представляет собой объект, где каждое числовое поле соответствует номеру вершины в массиве `vertex`. В каждом поле находится массив номеров других вершин, с которыми должна быть соединена данная вершина.

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 по теме «Стереометрия»

Для примера возьмём задание №27074 [5].



Заготовка шаблона имеет вид.

```
1 (function () {
2   retryWhileError(function () {
3     NAinfo.requireApiVersion(0, 2);
4
5     let paint1 = function (ctx) {
6     };
7
8     NATask.setTask({
9       text: 'Объем параллелепипеда ABCDA_1B_1C_1D_1 равен 9. Найдите объем
треугольной пирамиды ABCA_1.',
10      answers: 0,
11      author: ['Суматохина Александра']
12    });
13    NATask.modifiers.addCanvasIllustration({
14      width: 400,
15      height: 400,
```

```

16     paint: paint1,
17   });
18   }, 100000);
19 })();

```

1. Создадим объект класса `Parallelepiped` со случайной высотой, шириной и глубиной в заданном диапазоне.

```

1  (function () {
2    retryWhileError(function () {
3      NAinfo.requireApiVersion(0, 2);
4
5      let par = new Parallelepiped({
6        depth: sl(10, 50),
7        height: sl(10, 50),
8        width: sl(10, 50),
9      });
10
11     let paint1 = function (ctx) {
12     };
13
14     NATask.setTask({
15       text: 'Объем параллелепипеда ABCDA_1B_1C_1D_1 равен 9. Найдите
16       объем треугольной пирамиды ABCA_1.',
17       answers: 0,
18       author: ['Суматохина Александра']
19     });
20     NATask.modifiers.addCanvasIllustration({
21       width: 400,
22       height: 400,
23       paint: paint1,
24     });
25   }, 100000);
26 })();

```

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

```

1  (function () {
2    retryWhileError(function () {
3      NAinfo.requireApiVersion(0, 2);
4
5      let par = new Parallelepiped({
6        depth: sl(10, 50),
7        height: sl(10, 50),
8        width: sl(10, 50),
9      });

```

```

10
11     let camera = {
12         x: 0,
13         y: 0,
14         z: 0,
15         scale: 5,
16
17         rotationX: -Math.PI / 2 + Math.PI / 14,
18         rotationY: 0,
19         rotationZ: 2 * Math.PI / 3,
20     };
21
22     let point2DPar = par.verticesOfFigure.map((coord3D) =>
project3DTo2D(coord3D, camera));
23
24     autoScale(par.verticesOfFigure, camera, point2DPar, {
25         startX: -180,
26         finishX: 160,
27         startY: -160,
28         finishY: 160,
29         maxScale: 50,
30     });
31
32     point2DPar = par.verticesOfFigure.map((coord3D) =>
project3DTo2D(coord3D, camera));
33
34     let paint1 = function (ctx) {
35     };
36
37     NATask.setTask({
38         text: 'Объем параллелепипеда ABCDA_1B_1C_1D_1 равен 9.
Найдите объем треугольной пирамиды ABСA_1.',
39         answers: 0,
40         author: ['Суматохина Александра']
41     });
42     NATask.modifiers.addCanvasIllustration({
43         width: 400,
44         height: 400,
45         paint: paint1,
46     });
47     }, 100000);
48 })();

```

3. Перемещаемся в середину иллюстрации. Отрисовываем фигуру функцией **drawFigure**, передав в неё матрицу связей для параллелепипеда.

```

1 (function () {
2     retryWhileError(function () {
3         NAinfo.requireApiVersion(0, 2);
4
5         let par = new Parallelepiped({
6             depth: sl(10, 50),

```

```

7      height: sl(10, 50),
8      width: sl(10, 50),
9  });
10
11  let camera = {
12    x: 0,
13    y: 0,
14    z: 0,
15    scale: 5,
16
17    rotationX: -Math.PI / 2 + Math.PI / 14,
18    rotationY: 0,
19    rotationZ: 2 * Math.PI / 3,
20  };
21
22  let point2DPar = par.verticesOfFigure.map((coord3D) =>
project3DTo2D(coord3D, camera));
23
24  autoScale(par.verticesOfFigure, camera, point2DPar, {
25    startX: -180,
26    finishX: 160,
27    startY: -160,
28    finishY: 160,
29    maxScale: 50,
30  });
31
32  point2DPar = par.verticesOfFigure.map((coord3D) =>
project3DTo2D(coord3D, camera));
33
34  let paint1 = function (ctx) {
35    let h = 400;
36    let w = 400;
37    ctx.translate(h / 2, w / 2);
38    ctx.lineWidth = 2;
39    ctx.strokeStyle = om.secondaryBrandColors;
40    let strok = [5, 4];
41    ctx.drawFigure(point2DPar, [
42      [strok],
43      [0, 1],
44      [strok, 0, 1],
45      [0, 0, 0, 1],
46      [strok, 0, 0, 0, 1],
47      [0, 1, 0, 0, 0, 1],
48      [0, 0, 1, 0, 1, 0, 1],
49    ]);
50  };
51
52  NATask.setTask({
53    text: 'Объем параллелепипеда ABCDA_1B_1C_1D_1 равен 9.
Найдите объем треугольной пирамиды ABСA_1.',
54    answers: 0,
55    author: ['Суматохина Александра']

```

```

56         });
57         NATask.modifiers.addCanvasIllustration({
58             width: 400,
59             height: 400,
60             paint: paint1,
61         });
62     }, 100000);
63 })();

```

4. Далее вырезаем из условия значения и заменяем их данными из класса. Впишем ответ. Обособляем имена фигур в `$...$`. Добавляем буквы на вершины параллелепипеда. Добавляем модификаторы `NATask.modifiers.assertSaneDecision` (исключает нецелый ответ) и `NATask.modifiers.variativeABC(letter)` (заменяет все буквы в задании на случайные).

```

1  (function() {
2      retryWhileError(function() {
3          NAinfo.requireApiVersion(0, 2);
4
5          let par = new Parallelepiped({
6              depth: sl(10, 50),
7              height: sl(10, 50),
8              width: sl(10, 50),
9          });
10
11         let pyr = new Pyramid({
12             height: par.height,
13             baseArea: 0.5 * par.baseArea,
14         });
15
16         let camera = {
17             x: 0,
18             y: 0,
19             z: 0,
20             scale: 5,
21
22             rotationX: -Math.PI / 2 + Math.PI / 14,
23             rotationY: 0,
24             rotationZ: 2 * Math.PI / 3,
25         };
26
27         let point2DPar = par.verticesOfFigure.map((coord3D) =>
project3DTo2D(coord3D, camera));
28
29         autoScale(par.verticesOfFigure, camera, point2DPar, {
30             startX: -180,
31             finishX: 160,
32             startY: -160,
33             finishY: 160,
34             maxScale: 50,
35         });

```

```

36
37   point2DPar = par.verticesOffFigure.map((coord3D) =>
project3DTo2D(coord3D, camera));
38
39   let letter = ['A', 'B', 'C', 'D', '⊠D', '⊠A', '⊠B', '⊠C',];
40
41   let paint1 = function(ctx) {
42     let h = 400;
43     let w = 400;
44     ctx.translate(h / 2, w / 2);
45     ctx.lineWidth = 2;
46     ctx.strokeStyle = om.secondaryBrandColors;
47     let strok = [5, 4];
48     ctx.drawFigure(point2DPar, [
49       [strok],
50       [0, 1],
51       [strok, 0, 1],
52       [0, 0, 0, 1],
53       [strok, 0, 0, 0, 1],
54       [0, 1, 0, 0, 0, 1],
55       [0, 0, 1, 0, 1, 0, 1],
56     ]);
57
58     ctx.font = "25px liberation_sans";
59     point2DPar.forEach((elem, i) => ctx.fillText(letter[i],
elem.x, elem.y + ((i < point2DPar.length / 2) ? 15 : -10)));
60   };
61
62   NATask.setTask({
63     text: 'Объем параллелепипеда $ABCD A_1 B_1 C_1 D_1$ равен
$'+par.volume+'$. Найдите объем треугольной пирамиды $ABCA_1$.',
64     answers: par.volume/6,
65     author: ['Суматохина Александра']
66   });
67
68   NATask.modifiers.assertSaneDecimals();
69   NATask.modifiers.variativeABC(letter);
70
71   NATask.modifiers.addCanvasIllustration({
72     width: 400,
73     height: 400,
74     paint: paint1,
75   });
76   }, 100000);
77 })();

```


Заключение

За этот год был полностью покрыт открытый банк заданий ФИПИ по темам:

- Планиметрия — 26 шаблонов принято.
- Вектора — 18 шаблонов (10 принято, 8 на внутреннем рецензировании).
- Стереометрия — 56 шаблонов (7 принято, 49 на внутреннем рецензировании).
- Теория вероятности — 10 шаблонов на внутреннем рецензировании.
- Теория вероятности (повышенной сложности) — 11 шаблонов (1 принят 10 на внутреннем рецензировании).

В ядро проекта добавлены:

- Функции, упрощающие написание шаблонов по темам «Планиметрия» и «Стереометрия».
- Класс многогранников.
- Линейный проектор из $\mathbb{R}^3 \rightarrow \mathbb{R}^2$.

А также сокращён технический долг проекта.

Все добавленные в проект задания можно использовать для составления контрольных работ, проведения текущего контроля знаний учащихся, подготовки к ЕГЭ. [6]

В будущем планируется добавить в проект класс плоских геометрических фигур и использовать в заданиях по теме «Планиметрия» динамические изображения.

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] Федеральный институт педагогических измерений. – URL: <https://fipi.ru/ege/otkrytyy-bank-zadaniy-ege>
- [4] Единый государственный экзамен. – URL: https://ru.wikipedia.org/wiki/Единый_государственный_экзамен
- [5] Решу ЕГЭ - Сдам ГИА. – URL: <https://ege.sdamgia.ru/problem?id=27074>
- [6] Тренажёр "Час ЕГЭ". – URL: <https://math.vsu.ru/chas-ege/doc/index.html>

Приложение

```
1 (function() {
2   retryWhileError(function() {
3     NAinfo.requireApiVersion(0, 2);
4
5     let a = sl(2, 89);
6     let b = slKrome(a, 1, a - 1);
7
8     let vertices = window.latbukv.iz(4);
9
10    let angle = sl1() ? vertices.slice(0, 3).permuteCyclic(1) :
vertices.slice(0, 3).permuteCyclic(2);
11
12    genAssertZ1000(b / a, 'Кривой ответ');
13
14    let paint1 = function(ctx) {
15      ctx.lineWidth = 2;
16      ctx.strokeStyle = om.primaryBrandColors [0];
17
18      ctx.drawLine(10, 370, 390, 370);
19      ctx.drawLine(10, 370, 180, 200);
20      ctx.drawLine(180, 200, 390, 370);
21
22      //высота
23      ctx.lineWidth = 1.2;
24      ctx.drawLine(180 - 40, 200 - 33, 390, 370);
25      ctx.drawLine(180 - 40, 200 - 33, 10, 370);
26
27      //прямой угол
28      ctx.strokeStyle = om.secondaryBrandColors.iz();
29      ctx.drawLine(180 - 52, 200 - 13, 180 - 35, 200 - 1);
30      ctx.drawLine(180 - 35, 200 - 1, 180 - 23, 200 - 17);
31
32      //штрихи
33      ctx.drawLine(275 + 10, 210 + 90, 300, 200 + 90);
34      ctx.drawLine(80, 200 + 90, 105 - 10, 210 + 90);
35
36      ctx.font = "23px liberation_sans";
37      ctx.fillText(vertices[0], 10 - 5, 370 + 25);
38      ctx.fillText(vertices[1], 180, 200 - 10);
39      ctx.fillText(vertices[2], 390 - 10, 370 + 25);
40      ctx.fillText(vertices[3], 180 - 40, 200 - 40);
41
42    };
43
44    NATask.setTask({
45      text: 'В треугольнике $' + vertices.slice(0, 3).shuffle().join('') +
'$ $' +
46      vertices.slice(0, 2).shuffle().join('') +
```

```

47     '=' + vertices.slice(1, 3).shuffle().join('') + '$, $' +
vertices[0] + vertices[2] + '=' + a + '$, высота $' +
48     [vertices[0], vertices[3]].shuffle().join('') +
49     '$ равна $' + b + '$. Найдите $\sin \angle ' + angle.join('') +
'$.',
50     answers: b / a,
51     analys: '',
52   });
53   NATask.modifiers.addCanvasIllustration({
54     width: 400,
55     height: 400,
56     paint: paint1,
57   });
58   }, 1000);
59   })();
60   // 105

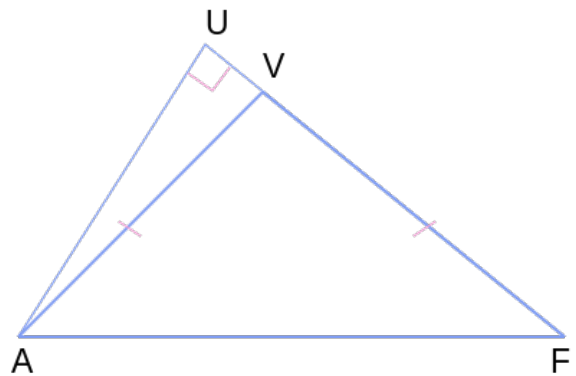
```

Листинг 3: 105.js

Примеры генерируемых задач 105.js

В треугольнике FAF $VA = VF$,
 $AF = 75$, высота AU равна 39. Най-
дите $\sin \angle VFA$.

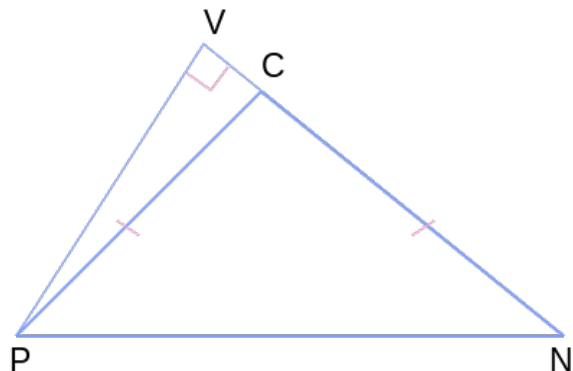
Ответ: 0,52



Задача. 1: Пример генерации задания с помощью 105.js

В треугольнике NCP $CP = CN$, $PN = 40$, высота VP равна 6. Найдите $\sin \angle CNP$.

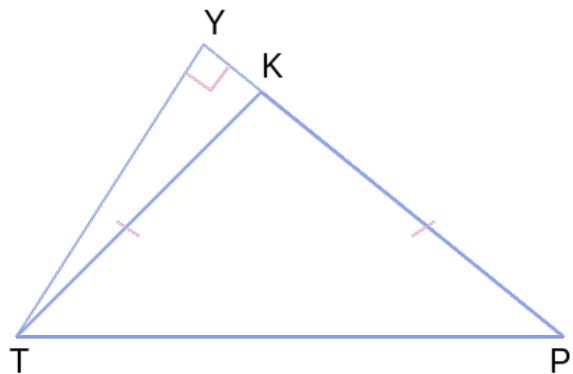
Ответ: 0,15



Задача. 2: Пример генерации задания с помощью 105.js

В треугольнике PKT $KT = PK$, $TP = 4$, высота TY равна 2. Найдите $\sin \angle KPT$.

Ответ: 0,5



Задача. 3: Пример генерации задания с помощью 105.js

```

1 (function() {
2   retryWhileError(function() {
3     NAinfo.requireApiVersion(0, 2);
4
5     let stroke = [4, 5];
6
7     let matrixConnections = {
8       0: [1, [3, stroke], 5],
9       2: [1, [3, stroke], 7],
10      4: [
11        [3, stroke],
12        [5, stroke],
13        [11, stroke]
14      ],
15      6: [1, 7, 9],

```

```

16      8: [5, [11, stroke], 13],
17      10: [7, 9, 15],
18      12: [
19          [11, stroke], 13, 15
20      ],
21      14: [9, 13, 15],
22  };
23
24  let par1 = new Parallelepiped({
25      depth: sl(10, 20),
26      height: sl(5, 20),
27      width: sl(10, 20),
28  });
29
30  let par2 = new Parallelepiped({
31      depth: par1.depth,
32      height: sl(5, 20),
33      width: slKrome(par1.width, 5, par1.width - 5),
34  });
35
36  let vertex3D =
37  par1.verticesOfFigure.concat(par2.verticesOfFigure.map((elem) =>
38  shiftCoordinate3D(elem, {
39      x: 0,
40      y: 0,
41      z: -0.5 * (par1.height + par2.height),
42  })));
43
44  vertex3D = vertex3D.map((elem) => shiftCoordinate3D(elem, {
45      x: 0,
46      y: 0,
47      z: 0.5 * par2.height,
48  })));
49
50  let camera = {
51      x: 0,
52      y: 0,
53      z: 0,
54      scale: 1,
55
56      rotationX: -Math.PI / 2 + Math.PI / 14,
57      rotationY: 0,
58      rotationZ: Math.PI / sl(10, 14),
59  };
60
61  let point2D = vertex3D.map((coord3D) => project3DTo2D(coord3D,
62  camera));
63
64  autoScale(vertex3D, camera, point2D, {
65      startX: -180,
66      finishX: 160,
67      startY: -160,

```

```

65     finishY: 160,
66     maxScale: 100,
67 });
68
69 point2D = vertex3D.map((coord3D) => project3DTo2D(coord3D, camera));
70 genAssert((point2D[4].x - point2D[8].x).abs() > 20);
71 genAssert((point2D[4].y - point2D[13].y).abs() > 50);
72 genAssert((point2D[12].x - point2D[14].x).abs() > 40);
73
74 let rand = sl1();
75
76 let paint1 = function(ctx) {
77     let h = 400;
78     let w = 400;
79     ctx.translate(w / 2, h / 2);
80     ctx.lineWidth = 2;
81     ctx.strokeStyle = om.secondaryBrandColors;
82     ctx.drawFigureVer2(point2D, matrixConnections);
83
84     if (point2D[4].x > point2D[8].x) {
85         let point = [point2D[4], point2D[5], point2D[8],
point2D[13]].mt_coordinatesOfIntersectionOfTwoSegments();
86         ctx.drawLine(point2D[5].x, point2D[5].y, point.x, point.y);
87     } else {
88         ctx.drawLine(point2D[4].x, point2D[4].y, point2D[5].x,
point2D[5].y);
89         let point = [point2D[4], point2D[11], point2D[8],
point2D[13]].mt_coordinatesOfIntersectionOfTwoSegments();
90         ctx.drawLine(point2D[4].x, point2D[4].y, point.x, point.y);
91     }
92
93     ctx.font = "20px liberation_sans";
94     ctx.signSegmentInMiddle(point2D[2].x, point2D[2].y, point2D[7].x,
point2D[7].y, par1.height, 10, 20);
95     ctx.signSegmentInMiddle(point2D[10].x, point2D[10].y,
point2D[15].x, point2D[15].y, par2.height, 10, 20);
96     ctx.signSegmentInMiddle(point2D[12].x, point2D[12].y,
point2D[15].x, point2D[15].y, par2.width, -10, 20);
97     ctx.signSegmentInMiddle(point2D[0].x, point2D[0].y, point2D[1].x,
point2D[1].y, par1.width, 18, 20);
98     ctx.signSegmentInMiddle(point2D[1].x, point2D[1].y, point2D[2].x,
point2D[2].y, par1.depth, 18, 20);
99 };
100 NATask.setTask({
101     text: 'Найдите ' + ['площадь поверхности', 'объём'][rand] +
102         ' многогранника, изображённого на рисунке все( двугранные углы -
прямые).',
103     answers: [par1.surfaceArea + par2.surfaceArea - 2 * par2.baseArea,
par1.volume + par2.volume][rand],
104 });
105 NATask.modifiers.addCanvasIllustration({
106     width: 400,

```

```

107         height: 400,
108         paint: paint1,
109     });
110 },
111     1000);
112
113 })();
114 //27193 25671 25673 25675 25677 25679

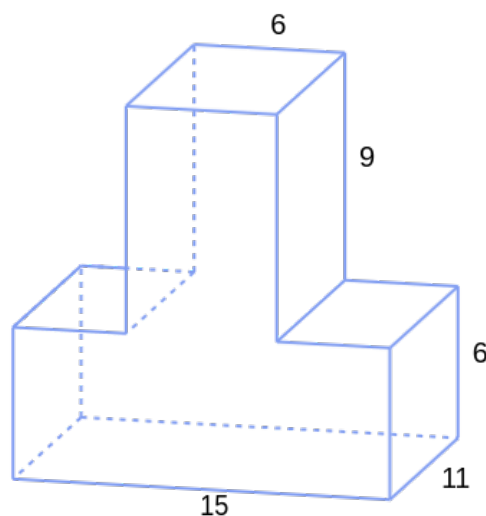
```

Листинг 4: 29193.js

Примеры генерируемых задач 27193.js

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

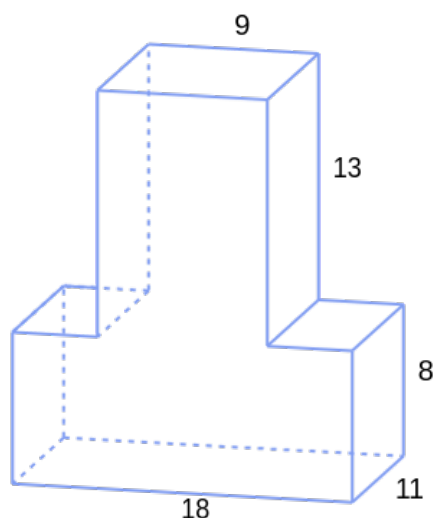
Ответ: 948



Задача. 4: Пример генерации задания с помощью 27193.js

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

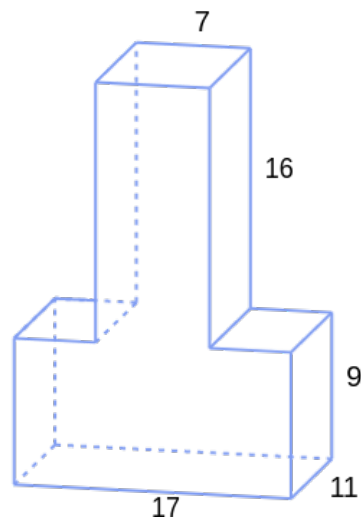
Ответ: 2871



Задача. 5: Пример генерации задания с помощью 27193.js

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

Ответ: 1454



Задача. 6: Пример генерации задания с помощью 27193.js

```

1 (function() {
2   retryWhileError(function() {
3     lx_declareClarifiedPhrase('сторона', 'основания');
4     NAinfo.requireApiVersion(0, 2);
5
6     let pyr = new RegularPyramid({
7       height: sl(20, 50),
8       baseSide: sl(20, 40),
9       numberSide: 4
10    });
11
12    pyr.verticesOfFigure.push({
13      x: 0,
14      y: 0,
15      z: pyr.verticesOfFigure[0].z
16    });
17
18    let question = [
19      [sklonlxkand('боковое ребро'), pyr.sideEdge],
20      [sklonlxkand('объём'), pyr.volume],
21    ].shuffle();
22    question.unshift([sklonlxkand('сторона основания'), pyr.baseSide]);
23
24    let camera = {
25      x: 0,
26      y: 0,
27      z: 0,
28      scale: 5,
29
30      rotationX: -Math.PI / 2 + Math.PI / 14,
31      rotationY: 0,
32      rotationZ: [1, 2].iz() * Math.PI / 3,
33    };

```

```

34
35   let point2DPyr = pyr.verticesOfFigure.map((coord3D) =>
project3DTo2D(coord3D, camera));
36
37   autoScale(pyr.verticesOfFigure, camera, point2DPyr, {
38     startX: -180,
39     finishX: 160,
40     startY: -160,
41     finishY: 160,
42     maxScale: 50,
43   });
44
45   point2DPyr = pyr.verticesOfFigure.map((coord3D) =>
project3DTo2D(coord3D, camera));
46
47   let letters = ['A', 'B', 'C', 'D', 'S', 'O'];
48
49   let strok = [5, 4];
50
51
52   let paint1 = function(ctx) {
53     let h = 400;
54     let w = 400;
55     ctx.translate(h / 2, w / 2);
56     ctx.lineWidth = 2;
57     ctx.strokeStyle = om.secondaryBrandColors;
58     ctx.drawFigure(point2DPyr, [
59       [1],
60       [strok, strok],
61       [1, strok, strok],
62       [1, 1, strok, 1, 0, strok],
63     ]);
64
65     ctx.font = "30px liberation_sans";
66     point2DPyr.forEach((elem, i) => ctx.fillText(letters[i], elem.x,
elem.y + ((i != point2DPyr.length - 2) ? 15 : -
67       10)));
68   };
69
70   NATask.setTask({
71     text: 'В правильной четырёхугольной пирамиде $SABCD$ с ' + 'основанием
$ABCD$ ' +
72       [question[0][0].ie + ' рав' + ['ен', 'на',
'но']][question[0][0].rod] + ' $' + question[0][1].pow(2).texsqrt(1) +
'$ ',
73     question[1][0].ie + ' рав' + ['ен', 'на', 'но']][question[1][0].rod]
+ ' $' + question[1][1].pow(2).texsqrt(1) + '$ '
74     ].shuffleJoin(', ') +
75     '. Найдите ' + question[2][0].ve + ' пирамиды.',
76     answers: question[2][1],
77     author: ['Суматохина Александра'],
78   });

```

```

79   NATask.modifiers.variativeABC(letters);
80   NATask.modifiers.multiplyAnswerBySqrt(13);
81   NATask.modifiers.allDecimalsToStandard(true);
82   NATask.modifiers.assertSaneDecimals();
83   NATask.modifiers.addCanvasIllustration({
84     width: 400,
85     height: 400,
86     paint: paint1,
87   });
88 },10);
89 })();
90 //https://ege314.ru/8-stereometriya-ege/reshenie-3011/
91 //3011

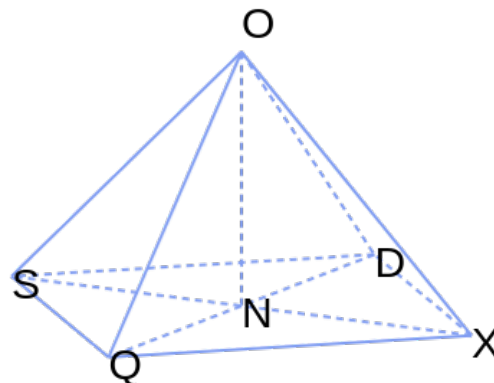
```

Листинг 5: 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.

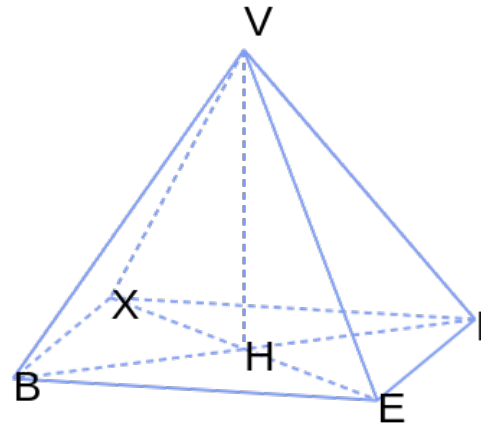
Ответ: 13689



Задача. 7: Пример генерации задания с помощью 3011.js

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

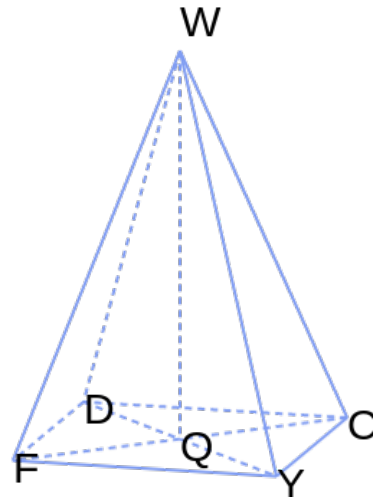
Ответ: 5346



Задача. 8: Пример генерации задания с помощью 3011.js

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

Ответ: 6720



Задача. 9: Пример генерации задания с помощью 3011.js

```

1 (function() {
2   retryWhileError(function() {
3     NAinfo.requireApiVersion(0, 2);
4
5     let angle = sl(2, 44);
6     let condition = [
7       ['большой', 45 + angle],
8       ['меньший', 45 - angle]
9     ].iz();
10
11     let paint1 = function(ctx) {
12       ctx.lineWidth = 2;
13
14       let angle = Math.PI/2.9;
15

```

```

16     ctx.strokeStyle = om.secondaryBrandColors.iz();
17     ctx.drawLine(10, 250, 390-8, 250);
18     let ver = ctx.drawLineAtAngle(10, 250, -angle, 200-25);
19     ctx.drawLineAtAngle(ver.x, ver.y, -angle+Math.PI/2, 350-20);
20     //штрихи
21     ctx.strokeInMiddleOfSegment(10, 250, (390-8)/2, 250, 10);
22     ctx.strokeInMiddleOfSegment(10, 250, 3*(390-8)/2, 250, 10);
23     //биссектриса
24     ctx.strokeStyle = om.primaryBrandColors [0];
25     let bis = ctx.drawLineAtAngle(ver.x, ver.y,
(-angle+Math.PI/2)+Math.PI/4, 160+2);
26     //медиана
27     ctx.strokeStyle = om.primaryBrandColors [1];
28     ctx.drawLine(ver.x, ver.y, (390-8)/2, 250);
29
30     ctx.strokeStyle = om.primaryBrandColors [0];
31     ctx.arcBetweenSegments([10, 250, ver.x, ver.y, bis.x, bis.y], 30);
32     ctx.arcBetweenSegments([390-8, 250, ver.x, ver.y, bis.x, bis.y], 38);
33
34 };
35
36 NATask.setTask({
37     text: 'Угол между биссектрисой и медианой прямоугольного треугольника, ' +
38         'проведёнными из вершины прямого угла, равен $' + angle + '~{\circ}$.'
39     'Найдите ' + condition[0] + ' угол прямоугольного треугольника. Ответ
дайте в градусах.',
40     answers: condition[1],
41     analys: '',
42 });
43 NATask.modifiers.addCanvasIllustration({
44     width: 400,
45     height: 400,
46     paint: paint1,
47 });
48 }, 1000);
49 })();
50 //2069

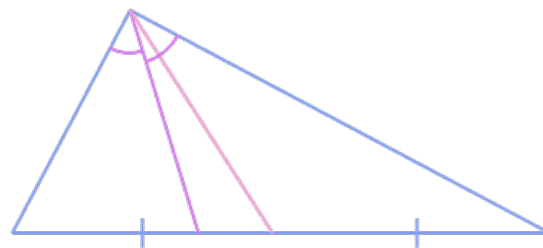
```

Листинг 6: 2069.js

Примеры генерируемых задач 2069.js

Угол между биссектрисой и медианой прямоугольного треугольника, проведёнными из вершины прямого угла, равен 38° . Найдите больший угол прямоугольного треугольника. Ответ дайте в градусах.

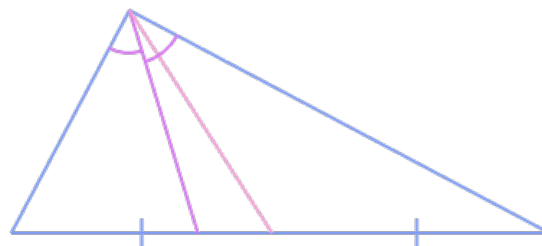
Ответ: 83



Задача. 10: Пример генерации задания с помощью 2069.js

Угол между биссектрисой и медианой прямоугольного треугольника, проведёнными из вершины прямого угла, равен 8° . Найдите меньший угол прямоугольного треугольника. Ответ дайте в градусах.

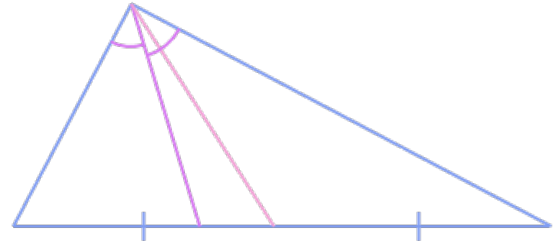
Ответ: 37



Задача. 11: Пример генерации задания с помощью 2069.js

Угол между биссектрисой и медианой прямоугольного треугольника, проведёнными из вершины прямого угла, равен 31° . Найдите меньший угол прямоугольного треугольника. Ответ дайте в градусах.

Ответ: 14



Задача. 12: Пример генерации задания с помощью 2069.js

```

1 (function() {
2   retryWhileError(function() {
3     NAinfo.requireApiVersion(0, 2);
4
5     let angle = sl(2, 89);
6
7     let vertices = window.latbukv.iz(6);
8
9     let rand = sl1();
10
11    let paint1 = function(ctx) {
12      ctx.lineWidth = 2;
13
14      let angle = -Math.PI / 3.2;
15      ctx.strokeStyle = om.secondaryBrandColors.iz();
16
17      let vertex = ctx.drawLineAtAngle(10, 370, angle, 400);
18      ctx.drawLine(10, 370, 390, 370);
19      ctx.drawLine(390, 370, vertex.x, vertex.y);
20
21      //Биссектрисы
22      let bisector1 = ctx.drawLineAtAngle(10, 370, angle / 2, 345);
23      let bisector2 = ctx.drawLineAtAngle(390, 370, Math.atan2(-370 +
24        vertex.y, -390 + vertex.x) / 2 - Math.PI / 2, 317);
25
26      //Углы
27      ctx.strokeStyle = om.primaryBrandColors[rand];
28      ctx.arcBetweenSegmentsCount([vertex.x, vertex.y, 10,
29        370].concat([bisector1.x, bisector1.y]), 30, 2);
30      ctx.arcBetweenSegmentsCount([bisector1.x, bisector1.y].concat([10,
31        370, 390, 370])), 40, 2);
32
33      ctx.strokeStyle = om.primaryBrandColors[rand];

```

```

31     ctx.arcBetweenSegments([10, 370, 390, 370].concat([bisector2.x,
bisector2.y]), 30);
32     ctx.arcBetweenSegments(([bisector2.x, bisector2.y].concat([390, 370,
vertex.x, vertex.y])), 40);
33
34     ctx.strokeStyle = om.primaryBrandColors[1 - rand];
35     ctx.arcBetweenSegmentsCount([bisector1.x, bisector1.y].concat([10,
370]).concat([bisector2.x, bisector2.y]).concat([390, 370]), 25, 3);
36
37     ctx.font = "23px liberation_sans";
38     ctx.fillText(vertices[0], vertex.x, vertex.y - 10);
39     ctx.fillText(vertices[1], 10 - 5, 370 + 20);
40     ctx.fillText(vertices[2], 390 - 20, 370 + 20);
41
42     ctx.fillText(vertices[3], bisector1.x, bisector1.y);
43     ctx.fillText(vertices[4], bisector2.x - 20, bisector2.y);
44
45     ctx.fillText(vertices[5], 210, 250);
46 };
47
48 NATask.setTask({
49     text: 'В треугольнике $' + vertices.slice(0, 3).join('') + '$ угол $'
+ vertices[0] + '$ равен $' + angle +
50     '~{\c} $, углы $' + vertices[1] + '$ и $' + vertices[2] + '$ -
острые, ' +
51     'биссектрисы $' + [vertices[1], vertices[3]].shuffleJoin() + '$ и
$' + [vertices[2], vertices[4]].shuffleJoin() +
52     '$ пересекаются в точке $' + vertices[5] + '$. Найдите угол $' +
vertices[2] + vertices[5] + vertices[1] +
53     '$. Ответ дайте в градусах.',
54     answers: 90 + 0.5 * angle,
55     analys: '',
56 });
57 NATask.modifiers.addCanvasIllustration({
58     width: 400,
59     height: 400,
60     paint: paint1,
61 });
62 }, 1000);
63 })();
64 //27764 628357 628475 47369 47371 47373 47375 47377 47379 47381 47383
47385 47387 47389 47391 47393 47395 47397 47399 47401 47403 47405 47407
47409 47411 47413 47415 47417 47419 47421 47423 47425 47427 47429 47431
47433 47435 47437 47439 47441 47443 47445 47447 47449 47451 47453

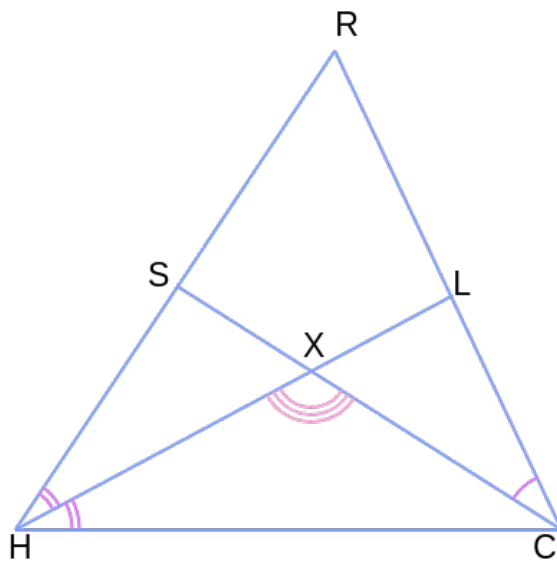
```

Листинг 7: 27764.js

Примеры генерируемых задач 27764.js

В треугольнике RHC угол R равен 67° , углы H и C – острые, биссектрисы LH и SC пересекаются в точке X . Найдите угол CXH . Ответ дайте в градусах.

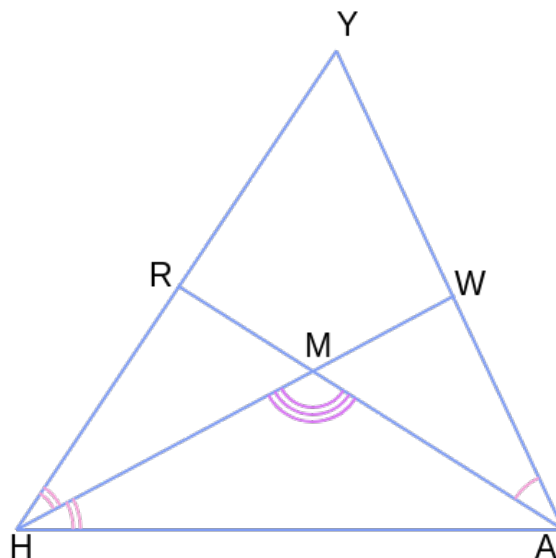
Ответ: 123,5



Задача. 13: Пример генерации задания с помощью 27764.js

В треугольнике YHA угол Y равен 45° , углы H и A – острые, биссектрисы HW и AR пересекаются в точке M . Найдите угол AMH . Ответ дайте в градусах.

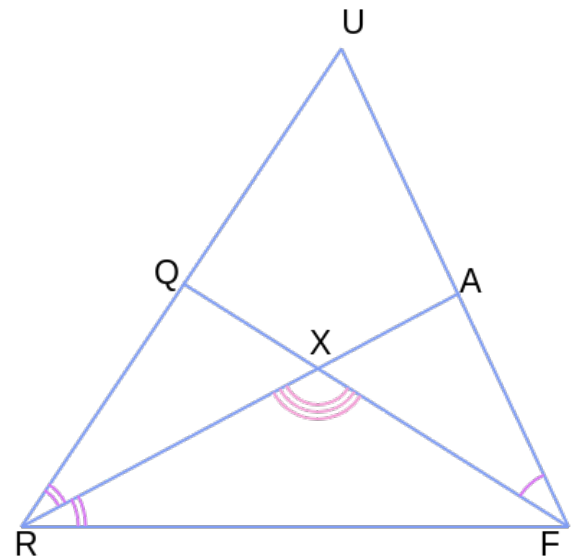
Ответ: 112,5



Задача. 14: Пример генерации задания с помощью 27764.js

В треугольнике URF угол U равен 7° , углы R и F – острые, биссектрисы AR и FQ пересекаются в точке X . Найдите угол FXR . Ответ дайте в градусах.

Ответ: 93,5



Задача. 15: Пример генерации задания с помощью 27764.js

```

1 (function() {
2   retryWhileError(function() {
3     NAinfo.requireApiVersion(0, 2);
4
5     let a = sl(2, 89);
6
7     let vertices = om.latbukv.iz(5);
8
9     let trapezoid = [vertices[0], vertices[1], vertices[4],
10    vertices[3]].randomReverse();
11
12    let paint1 = function(ctx) {
13      ctx.lineWidth = 2;
14      ctx.strokeStyle = om.secondaryBrandColors.iz();
15
16      ctx.drawLine(100, 80, 350, 80);
17      ctx.drawLine(10, 320, 270, 320);
18      ctx.drawLine(100, 80, 10, 320);
19      ctx.drawLine(270, 320, 350, 80);
20
21      ctx.strokeStyle = om.primaryBrandColors[0];
22      let middle = coordinatesMiddleOfSegment(270, 320, 350, 80);
23      ctx.drawLine(10, 320, middle[0], middle[1]);
24
25      ctx.strokeStyle = om.primaryBrandColors[1];
26      ctx.strokeInMiddleOfSegment(350, 80, middle[0], middle[1], 10);
27      ctx.strokeInMiddleOfSegment(270, 320, middle[0], middle[1], 10);
28
29      ctx.font = "23px liberation_sans";
30      ctx.fillText(vertices[0], 100, 80 - 10);
31      ctx.fillText(vertices[1], 350, 80 - 10);
32      ctx.fillText(vertices[2], 270, 320 + 20);
33      ctx.fillText(vertices[3], 10, 320 + 20);
34      ctx.fillText(vertices[4], middle[0] + 5, middle[1] + 5);
35    };
36
37    NATask.setTask({
38      text: 'Площадь параллелограмма $' + vertices.slice(0,
39    4).permuteCyclic(sl(0, 3)).join('') + '$ равна $' + a +
40    '$. Точка $' + vertices[4] + '$ - середина стороны $' +
41    vertices.slice(1, 3).shuffleJoin() +
42    '$. Найдите площадь трапеции $' + trapezoid.permuteCyclic(sl(0,
43    3)).join('') + '$.',
44      answers: a * 0.75,
45    });
46    NATask.modifiers.addCanvasIllustration({
47      width: 400,
48      height: 400,
49      paint: paint1,
50    });
51  });
52 }());

```

```

48     }, 1000);
49   })();
50   //92

```

Листинг 8: 92.js

Примеры генерируемых задач 92.js

```

1  (function() {
2    retryWhileError(function() {
3
4      let cube = sl(2, 10).pow(3);
5      let radiusNew = cube;
6      let radiuses = [];
7
8      while (cube > 0) {
9        radiuses.push(slKrome(radiuses, 2, cube));
10       cube -= radiuses[radiuses.length - 1];
11     }
12     genAssert(radiuses.length < 5 && radiuses.length > 1, 'radiusNew');
13
14     let radiusView = radiuses.map((elem) => elem.texcbirt(1));
15     labelline:texcbirt
16
17     let paint1 = function(ctx) {
18       ctx.strokeStyle = om.secondaryBrandColors;
19       ctx.lineWidth = 2;
20       //шаг 1
21       ctx.drawArc(200, 200, 150, 0, Math.PI * 2, true);
22
23       // Внешняя окружность
24       ctx.drawEllipse(200, 200, 20, 150, Math.PI / 2, 1.5 * Math.PI,
25       Math.PI / 2);
26
27       ctx.setLineDash([5, 5]);
28       ctx.drawEllipse(200, 200, 20, 150, Math.PI / 2, Math.PI / 2, 1.5 *
29       Math.PI);
30
31       ctx.strokeStyle = om.primaryBrandColors.iz();
32       ctx.drawLine(200, 200, 200, 50);
33     };
34
35     NAinfo.requireApiVersion(0, 2);
36     NATask.setTask({
37       text: 'Радиусы ' + ['двух', 'трёх', 'четырёх'][radiuses.length - 2] +
38       ' шаров равны ' + ('$' + radiusView.join(
39         '$, $') + '$') + '. Найдите радиус шара, объем которого равен сумме их
40       объемов.',
41       answers: Math.cbrt(radiusNew),
42       authors: ['Суматохина Александра'],
43     });

```

```
39     chas2.task.modifiers.addCanvasIllustration({
40         width: 300,
41         height: 250,
42         paint: paint1,
43     });
44 }, 1000);
45 })();
46 //27125 75307 75309 75311 75313
```

Листинг 9: 27125.js

Примеры генерируемых задач 27125.js