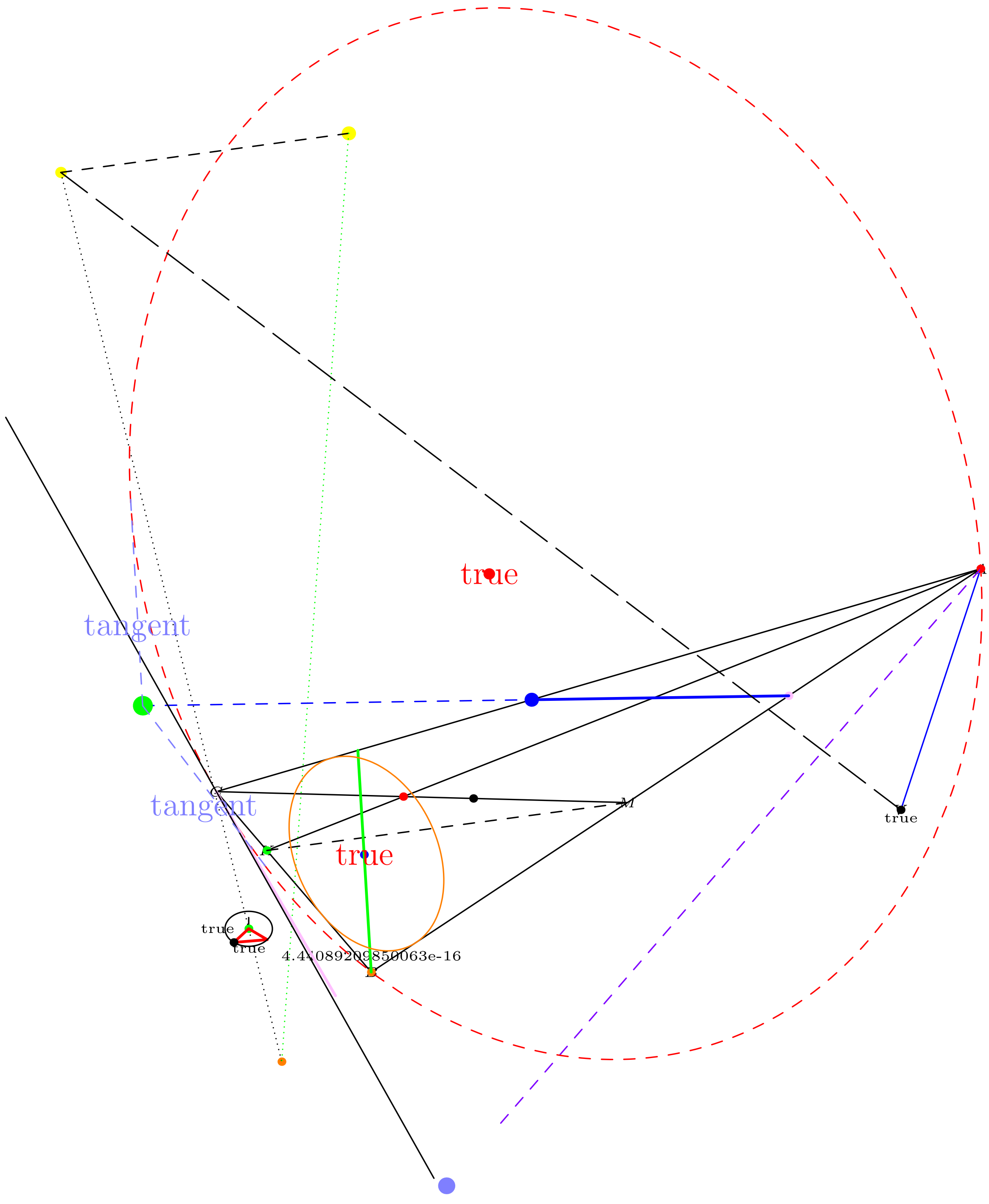
КРАТКОЕ ОПИСАНИЕ МОДУЛЯ GEOMETRY3



Автор: *Александр Калиев*

Проект на GitHub

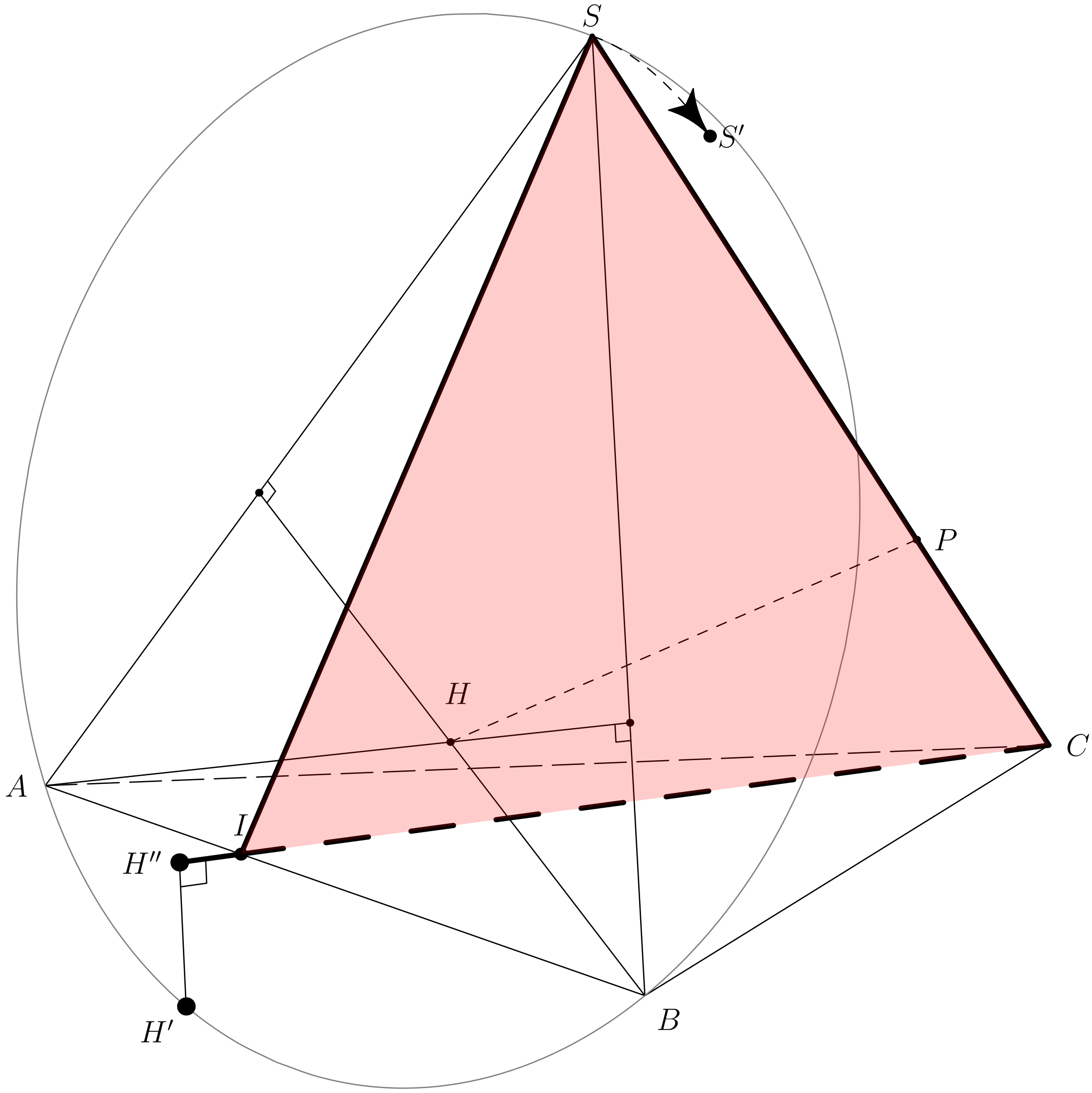
зима 2024

Содержание

Действия с модулем geometry3

- Нарисуем произвольный тетраэдр
- Отметим точку P, делящую в отношении $2:1$
- Проведём HP, где H – ортоцентр противолежащей грани SAB
- Опишем окружность около SAB
- Отразим ортоцентр относительно середины AB
- Спроецируем прямую CH' на грань ABC
- Отметим плоскость SIC красным цветом
- Затем повернём $\triangle SIC$ относительно оси IC на $20°$, и отметим точку S'.

Понятно, что чертёж очень нагромождён, но это была лишь демонстрация некоторой функциональности модуля.



1 Точки

В качестве точек используется *triple*. Отмечу также, что тройка применяется также и для векторов, что ничуть нам не мешает на ранней стадии развития модуля.

▷ *real distance(triple A, triple B)* возвращает длину отрезка AB.

2 Плоскости

Плоскости *plane* являются структурой модуля. Плоскость можно задать:

1. По 3-м точкам
2. По вектору нормали и точке
3. По 2 прямым *line3*
4. По треугольнику

```
plane p1 = plane(A,B,C);
plane p2 = plane(v, P);
plane p3 = plane(l1, l2);
plane p4 = plane(t);
```

Структура *plane* хранит в себе 4 вещественных числа a,b,c,d и одну тройку-вектор n. Как известно, плоскость можно задать уравнением вида $Ax + By + Cz + D = 0$; 4 вещественных числа структуры с этой ролью справляются довольно успешно. Также известно, что уравнение плоскости тесно связано с вектором нормали. При создании вычисляется вектор нормали n, причём он укорачивается до длины 1, а первая его координата делается *положительной* (для определённости).

▷ Метод плоскости *real calculate(triple P)* вычисляет значение для точки P, подставляя её координаты в уравнение плоскости. Таким образом, можно проверить, находится точка в плоскости или нет.

▷ *bool inplane(triple P)* проверяет, находится точка в плоскости или нет.

▷ Плоскости можно сравнивать: мы однозначно определили вектор n и поэтому можем сравнивать 2 плоскости, заданные разными точками, на равенство.

▷ Функция *real distance(triple P, plane p)* возвращает число – расстояние от точки до плоскости.

▷ Функция *triple projection(plane p, triple P)* возвращает точку-проекцию точки P на плоскость p.

3 Прямые

Прямые *line3* являются структурой модуля. В основном, прямая задаётся:

1. По 2 точкам
2. По точке и направляющему вектору

```
line3 l1 = line3(A,B) // по 2 точкам
line3 l1_2 = line3(A,B,false) // то же самое, что и l1
line3 l2 = line3(P,v,true) // по точке и вектору
```

Прямые создавались для того, чтобы искать пересечения.

```
line3 l1 = line3(A,B);
line3 l2 = line3(C,D);
triple P = intersectionpoint(l1,l2);
```

Прямая имеет 2 полезных метода: *getLine()* и *getBase()*: первый возвращает *path3* как бы бесконечной прямой, второй возвращает *path3* параметров, которые мы передали при создании прямой. Например, для кода

```
line3 l = line3(A,B);
draw(l.getBase());
```

будет нарисован отрезок AB. Для задания по вектору будет нарисован отрезок $(P,\ P{+}v)$.

Перечислим функции, связанные с прямыми:

- *bool inplane(line3 l, plane p)* возвращает *bool* и показывает, находится ли прямая в плоскости
- *bool parallel(line3 l1, line3 l2* показывает, параллельны ли прямые
- *bool crossing(line3 l1, line3 l1)* показывает, скрещиваются ли прямые
- *plane plane(line3 l1, line3 l2* формирует общую плоскость для ℓ_1 и ℓ_2. В случае, если такой нет, программа бросит исключение.
- *triple intersectionpoint(line3 l1, line3 l2, fuzz=-1)* ищет точку пересечения прямых ℓ_1, ℓ_2. *fuzz* является параметром погрешности, его рекомендуется изменять только в крайнем случае.
- *triple intersectionpoint(line3 l, plane p, real fuzz=-1)*. Аналогично, только между прямой и плоскостью.
- *line3 projection(plane p, line3 l)* проецирует прямую ℓ на плоскость p.
- *line3 raiseperpendicular(plane p, triple P)* восставляет перпендикуляр к p из точки P.

4 Треугольники

Треугольники *triangle3* являются структурой модуля и содержат в себе некоторую функциональность. Треугольник задаётся по 3 точкам A,B,C. Сразу перечислим основные методы треугольника:

```
// инициализация точек K,L,M...

triangle3 t = triangle3(K,L,M);

// t.A = K, t.B = L, t.C = M

t.A; t.B; t.C;  // обращение к вершинам

t.a();
t.b(); // длины сторон
t.c();

t.alpha();
t.beta(); // углы
t.gamma();

t.area(); // площадь
t.perimeter(); // периметр
```

Отметим также, что ко всем полям и точкам треугольника можно обращаться и по полю: *t.a, t.b, и так далее.*

Перечислим функции, связанные с треугольником и его геометрией:

```
centroid(t);
orthocenter(t);
circumcenter(t);
incenter(t);
circumcircle(t);
incircle(t);
median(t, t.A);
height(t, t.A);
bisector(t, t.A);
heightpoint(t, t.A);
bisectorpoint(t, t.A);
```

Как уже было сказано, по треугольнику можно задать плоскость.

В отличие от других структур, треугольник можно рисовать без вызова методов: *draw(t, red+1bp+dashed+opacity(0.7))*. Будет нарисован только контур, но это может измениться.

5 Окружности

Окружность *circle3* также создана в модуле. Её можно задать по центру, радиусу и нормали. По умолчанию нормаль – это вектор (0,0,1).

```
circle3 c = circle3((1,2,3), 4.5, (3,5,6));
```

К параметрам окружности можно обращаться по полю: *c.n, c.r,* $c.C$. Метод $c.getPlane()$ вернёт Вам плоскость, в которой находится окружность c. Метод $getPath()$ вернёт $path3$ для заданной окружности.

- ▷ *bool inplane(circle3 c, plane p)* проверяет, находится ли c в плоскости p.
- ▷ *bool oncircle(circle3 c, triple P)* проверяет, лежит ли точка P на окружности c.
- ▷ *line3 tangent(circle3 c, triple P)* вернёт прямую, которая является касательной к c, если точка P лежит **на окружности**.
- ▷ *triple[] tangents(circle3 c, triple P)* как правило возвращает массив из 2 точек касания для окружности c и точки P, если точка **не на окружности**.

6 Отметки

- ▷ *void markrightangle(triple A, triple B, triple C, real size = 1, pen p = currentpen, pen fillpen = nullpen, light light = currentlight)* отмечает угол ABC как прямой.
- ▷ *void markangle(triple A, triple B, triple C, int n = 1, real radius = 1, real space = 1, pen p = currentpen, pen fillpen = nullpen, light light = currentlight)* помечает угол ABC n засечками. Здесь *radius, space* являются лишь коэффициентами.

Отмечу, что на данный момент размер отметок не зависит от размера, поэтому часто придётся сильно изменять *radius, space* в зависимости от размера чертежа. Например, у меня в примере из начала для *markrightangle* присвоено *size=3*.

7 Трансформации

В модуле реализована лишь одна трансформация, однако вместе с ней существуют встроенные в *Asymptote* трансформации. Перечислю лишь некоторые:

- ▷ (модуль) *scale3(real k=1.0, triple P)*: гомотетия с центвом в точке P и коэффициентом k. Для центральной симметрии относительно P, очевидно, достаточно $k = -1$.
- ▷ *reflect(triple u, triple v, triple w)* отражает относительно плоскости uvw.
- ▷ *rotate(real d, triple k, triple m)* вращает относительно оси km на d градусов.

8 Пример из начала

Мы готовы разобрать код из самого первого примера.

```
settings.render = 0; // векторная графика (до импортов!)

import three;
import geometry3;
size(15cm);

triple A=(6,0,0),B=(6,6,0),C=(0,6,0),S=(4,4,6); // задаём вершины

// отрисовка SABC (шаг 1)
draw(A--B^^B--C);
draw(A--C, longdashed);
draw(S--A);
draw(S--B);
draw(S--C);
label("$A$", A, 2W);
label("$B$", B, 2SE);
label("$C$", C, 2E);
label("$S$", S, N);

// отметка точки (шаг 2)
triple P = relpoint(S--C, 2/3);
dot(P);
label("$P$", P, 2E);

// отметим ортоцентр и проведём HP (шаг 3)
triangle3 t = triangle3(S,A,B);
triple H = orthocenter(t);
draw(H--P, dashed);
label("$H$", H, S); dot(H);

// поясняем, что это ортоцентр
triple A1 = heightpoint(t, A), B1 = heightpoint(t, B);
markrightangle(A,A1,B,size=3);
markrightangle(S,B1,B,size=3);
draw(height(t, A)); dot(A1);
draw(height(t, B)); dot(B1);

// описываем окружность (шаг 4)
circle3 c = circumcircle(t);
draw(c.getPath(), grey);

// отражаем H (шаг 5)
triple M = midpoint(A--B);
```

```
triple H_ = scale3(-1, M)*H;
dot(H_, black+7bp);
label("$H'$", H_, 2SW);

// проецируем (шаг 6)
triangle3 t2 = triangle3(H_, C, S);
triple H__ = projection(plane(A,B,C), H_);
label("$H''$", H__, 2W);
dot(H__, black+7bp);
triple I = intersectionpoint(C--H__, A--B);
draw(H__--I, black+2bp);
draw(I--C, black+2bp+dashed);
draw(H_ -- H__);
dot(I, black+5bp);
label("$I$", I, 2N);
markrightangle(H_, H__, C, size=5);

// отмечаем плоскость (шаг 7)
surface s1 = surface(S--I--C--cycle);
draw(S--I, black+2bp);
draw(S--C, black+2bp);
draw(s1, red+opacity(0.2));

// поворачиваем и отмечаем (шаг 8)
triple S_ = rotate(20, I, C)*S;
dot("$S'$", S_, black+5bp);
triple T = heightpoint(triangle3(I,S,C), S);
draw(arc(T, S, S_), dashed, Arrow3(HookHead2, 15));
```