Contents

Introduction

Before using this package, make sure, that you have this settings:

```
settings.outformat = "pdf";
settings.render = 0;
settings.prc = false;
```

and specified size of picture by `size3`. Also, you have to wrap your code into function (say `main`) and put `with_geometry3d(main);` after `main` function ends.

Objects types list

The package *geometry3d.asy* is the extension of the module *geometry.asy*. Basically, this package provides you a tools to creare a really nice 3D pictures in solid geometry.

Here is all types, defined in this module
`basis3` – a 3D ray
`curve3` – a 3D ray
`ray3` – a 3D ray
`vector3` – a 3D vector
`line3` – a 3D line
`planeLine3` – a finite line on the given plane
`circle3` – a 3D circle
`plane3` – a plane
`sphere3` – a sphere

Temp: all functions

```
void drawAllObjects();
```

this function draws all objects on the scene with front-back feature and is called by default in function `with_geometry3d`.

```
void withGeometry3d(void main());
```

this function is meant to be ending of your programm, executing essential function for drawing figures properly.

```
void add2dFrame();
```

add 2D frame in order to be able to draw a 2D figures

```
void drawCurve(picture pic=currentpicture, curve3 curve,
pen frontpen=currentpen, pen backpen=currentpen+dashed);
```

draw `curve` with pens `frontpen` and `backpen` respectively.

```
circle3 circle3(triple A, triple B, triple C);
```

returns circumcircle of triangle ABC.

```
circle3 incircle3(triple A, triple B, triple C);
```

returns incircle of triangle ABC.

```
transform3 orthogonalproject(plane3 p);
```

returns `transform3`, which projects in direction of normal to the `plane p`.

```
triple foot3(triple A, line3 l);
```

return the foot of the perpendicular dropped from point A onto the line l.

```
triple foot3(triple A, plane3 p);
```

return the foot of the perpendicular dropped from point A to the plane p.

```
void markrightangle3(triple A, triple B, triple C, real
n=5, pen p=currentpen);
```

marks right angle $\angle ABC$ with `pen p`, size of `real n`.

```
real distance3(triple A, triple B);
```

returns distance between two points A and B.

```
triple midpoint3(triple A, triple B);
```

returns the midpoint of segment AB.

```
basis3 get_basis(projection P = currentprojection);
```

returns the basis of the `projection P` formed from vectors $\vec{x} =$ `P.camera`, $\vec{y} = \vec{x} \times \vec{u}$, $\vec{z} = \vec{x} \times \vec{y}$, where $\vec{u} =$ `P.up`.

```
triple calcCoordsInBasis(basis3 basis, triple A);
```

returns coordinates of point A (which coordinates are given in standart basis $\{\vec{x}, \vec{y}, \vec{z}\}$) in basis `basis`.

```
triple changeBasis(basis3 basis1, basis3 basis2, triple
A);
```

returns coordinates of point A (which coordinates are given in basis `basis1`) in basis `basis2`.

```
pair project3(triple A);
```

returns 2D-coordinates (x', y') of `triple A` as if it was drawn as a plain point A' with coordinates (x', y').

`WARNING!` It won't work unless you specified size of image with `size3`.

```
path project3(path3 p);
```

returns 2D-path formed from `project3(node)` for each node of nodes of `path3 p`.

```
void markangle3(picture pic = currentpicture, Label L =
"", int n = 1, real radius = 0, real space = 0, explicit
triple A, explicit triple B, explicit triple C, pair align
= dir(1), arrowbar3 arrow3 = None, pen p = currentpen,
filltype filltype_ = NoFill, margin margin = NoMargin,
marker marker = nomarker);
```

marks angle $\angle ABC$ with `pen p`, filled with `filltype_`, drawing arrow with `arrow3`.

```
bool collinear3(triple A, triple B, triple C);
```

returns `true` if points A, B, C are collinear, otherwise it will return `false`.

```
circle3 Circle(triple C, triple A, triple normal=Z);
```

returns circle with center at C and normal `normal`, passing through point A.

```
line3 parallel(line3 a, triple A);
```

returns line, which is parallel to given line a and passing through point A.

```
bool isIntersecting(line3 a, plane3 s, bool inf=true);
```

returns `true` if line a and plane s intersect, otherwise – `false`. If `inf=false`, then plane s is not considered infinite.

```
triple intersectionpoint(line3 a, plane3 s, bool
inf=true);
```

returns the intersection point of line a and plane s if they intersect, otherwise function aborts the program. If `inf=false`, then plane s is not considered infinite. We might use function `intersectionpoint(path3 p, surface s)`, considering line as a very long straight `path3` and plane as a real wide `surface`, but it takes a pretty long time to calculate this. So, we will calculate this by ourselves using algebra.
We define line as $\langle x, y, z\rangle = \langle x_0, y_0, z_0\rangle + \vec{v} \cdot t, t \in \mathbb{R}$ and plane as $Ax + By + Cz + D = 0$. Let $\vec{v} = \langle x_v, y_v, z_v\rangle$, then we can write a system of linear equations:

$$\begin{cases} x = x_0 + x_v \cdot t \\ y = y_0 + y_v \cdot t \\ z = z_0 + z_v \cdot t \\ Ax + By + Cz + D = 0 \end{cases} \iff \begin{cases} x - x_v \cdot t = x_0 \\ y - y_v \cdot t = y_0 \\ z - z_v \cdot t = z_0 \\ Ax + By + Cz = -D \end{cases} \tag{1}$$

The triple (x, y, z) of the solution (x, y, z, t) of that system defines the desired intersection point (we assume, that this point exists and is unique, because otherwise program would stopped a long ago). We can easily rewrite (1) using marices:

$$\begin{bmatrix} 1 & 0 & 0 & -x_v \\ 0 & 1 & 0 & -y_v \\ 0 & 0 & 1 & -z_v \\ A & B & C & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ t \end{bmatrix} = \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ -D \end{bmatrix}$$

Then we solve this matrix equation (using embedded function) and get the desired result which is the intersection point.

```
line3 invertpoint(pair A, projection P=currentprojection);
```

returns line, which contains point A and has the vector `P.camera` as its direction vector. This function will be still working if A has a type `triple`.

```
triple invert3(pair A, line3 a);
```

returns `triple` representation of `pair A`, where resulting `triple` point lies on the line a.
Basically, it's the inverse of function `project3`.

```
triple getpointX(real x, line3 a);
```

For the type `line3` are available functions `getpointX`, `getpointY`, `getpointZ`, which calculate the rest of the coordinates of the point on the given line by one given coordinate.

```
triple getpointXY(real x, real y, plane3 a);
```

For the type `plane3` are defined analogical functions `getpointXY`, `getpointYZ`, `getpointXZ`, which calculate the last one of the coordinates of the point on the given plane by two given coordinates.

```
bool isIntersecting(line3 a, line3 b);
```

returns `true` if lines a and b intersect.

```
bool isSkew(line3 a, line3 b);
```

returns `true` if lines a and b are skew.

```
bool isParallel(line3 a, line3 b);
```

returns `true` if lines a and b are parallel.

```
triple intersectionpoint(line3 a, line3 b);
```

returns the intersection point of the lines a and b.

The type `line3`

The type `sphere3`

Represent sphere `sphere(C,r);` as a circle `Circle(project3(C),r);` from package `graph`.