graph.js => graph.ts

export type TWIN2D = {

LEFT: number;

BOTTOM: number;

WIDTH: number;

HEIGHT: number;

}

export type TWIN3D = TWIN2D & {

CAMERA: Point;

CENTER: Point;

//class можно использовать в качестве типа данных

}

export type TWIN = TWIN2D | TWIN3D;

export type TGraph = {

id?:string;

width?:number;

height?:number;

WIN:TWIN;

callbacks:{

wheel: (event: WheelEvent) => void;

mousemove: (event: MouseEvent) => void;

mouseup: () => void;

mousedown: () => void;

mouseout: ()=> void;

}

}

class Graph {

private canvas:HTMLCanvasElement;

private context: CanvasRenderingContext2D;

private WIN:TWIN;

private PI2= Math.pi \* 2;

constructor({id, width:500, height = 500, WIN, callbacks}:TGraph){

if (id){

this.canvas = document.getElementById(id) as HTMLCanvasElement;

//as необходимо использовать как можно меньше

}else{

this.context = this.canvasgetContext(‘2D’) as CanvasRenderingContext2D;

this.WIN = WIN;

…

}

//каждый метод класса должен иметь типа возвращаемого значения

xs(n:number):number {…}

polygon(points: Omit < Point, ‘z’>[], color = ‘#00ff00’): void{

//…

}

}

Point.ts

class Point {

x:number;

y:number;

z:number;

constructor(x=0,y=0,z=0){

this.x = x;

this.y = y;

this.z = z;

}

}

//Также с остальными

Light.ts

class Light extends Point{

lumen: number;

constructor(x:number, y:number, z:number, lumen = 1500){

super(x,y,z);

this.lumen = lumen;

}

}

Polygon.ts

export type TRGB = {

r:number;

g:number;

b:number;

}

export enum Edistance{

distance = ‘distance’;

lumen = ‘lumen’;

}

class Polygon {

points: number[];

color:TRGB;

center = new Point();

[EDistance.distance]:number;

[EDistance.lumen]:number;

В math3D сделать enum3D в который записать все преобразования

Surface.ts

export type TAnimation = {

method: ETransform;

value:number;

center: Point;

}

class Surface {

points: Point[];

edges: Edge[];

polygons:Polygon[];

center:Point;

animations:TAnimation[];

}

setAnimation(method:ETransform, value:number, center?: Point):void{}

Math3D.ts

type TMath3D = {

WIN: TWIN3D;

}

type TMatrix = number[][];

type TVector = number[];

type TShadow = {

isShadow:Boolean;

dark?:number;

}

export enum Etransform{

zoom = ‘zoom’;

move = ‘move’;

rotateOx = ‘rotateOx’;

rotateOy = ‘rotateOy’;

rotateOz = ‘rotateOz’;

}

[ETransform.zoom](delta:number): TMatrix{};

// И так со всеми

math.calcDistance(surface, WIN.CAMERA, Edistance.distance){}