## **Làm code trái tim HTML cơ bản**

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>

<HEAD>

<TITLE> Heart </TITLE>

<META NAME="Generator" CONTENT="EditPlus">

<META NAME="Author" CONTENT="">

<META NAME="Keywords" CONTENT="">

<META NAME="Description" CONTENT="">

<style>

html, body {

height: 100%;

padding: 0;

margin: 0;

background: black;

}

canvas {

position: absolute;

width: 100%;

height: 100%;

}

</style>

</HEAD>

<BODY>

<canvas id="pinkboard"></canvas>

<script>

/\*

\* Settings

\*/

var settings = {

particles: {

length: 500, // maximum amount of particles

duration: 2, // particle duration in sec

velocity: 100, // particle velocity in pixels/sec

effect: -0.75, // play with this for a nice effect

size: 30, // particle size in pixels

},

};

/\*

\* RequestAnimationFrame polyfill by Erik Möller

\*/

(function(){var b=0;var c=["ms","moz","webkit","o"];for(var a=0;a<c.length&&!window.requestAnimationFrame;++a){window.requestAnimationFrame=window[c[a]+"RequestAnimationFrame"];window.cancelAnimationFrame=window[c[a]+"CancelAnimationFrame"]||window[c[a]+"CancelRequestAnimationFrame"]}if(!window.requestAnimationFrame){window.requestAnimationFrame=function(h,e){var d=new Date().getTime();var f=Math.max(0,16-(d-b));var g=window.setTimeout(function(){h(d+f)},f);b=d+f;return g}}if(!window.cancelAnimationFrame){window.cancelAnimationFrame=function(d){clearTimeout(d)}}}());

/\*

\* Point class

\*/

var Point = (function() {

function Point(x, y) {

this.x = (typeof x !== 'undefined') ? x : 0;

this.y = (typeof y !== 'undefined') ? y : 0;

}

Point.prototype.clone = function() {

return new Point(this.x, this.y);

};

Point.prototype.length = function(length) {

if (typeof length == 'undefined')

return Math.sqrt(this.x \* this.x + this.y \* this.y);

this.normalize();

this.x \*= length;

this.y \*= length;

return this;

};

Point.prototype.normalize = function() {

var length = this.length();

this.x /= length;

this.y /= length;

return this;

};

return Point;

})();

/\*

\* Particle class

\*/

var Particle = (function() {

function Particle() {

this.position = new Point();

this.velocity = new Point();

this.acceleration = new Point();

this.age = 0;

}

Particle.prototype.initialize = function(x, y, dx, dy) {

this.position.x = x;

this.position.y = y;

this.velocity.x = dx;

this.velocity.y = dy;

this.acceleration.x = dx \* settings.particles.effect;

this.acceleration.y = dy \* settings.particles.effect;

this.age = 0;

};

Particle.prototype.update = function(deltaTime) {

this.position.x += this.velocity.x \* deltaTime;

this.position.y += this.velocity.y \* deltaTime;

this.velocity.x += this.acceleration.x \* deltaTime;

this.velocity.y += this.acceleration.y \* deltaTime;

this.age += deltaTime;

};

Particle.prototype.draw = function(context, image) {

function ease(t) {

return (--t) \* t \* t + 1;

}

var size = image.width \* ease(this.age / settings.particles.duration);

context.globalAlpha = 1 - this.age / settings.particles.duration;

context.drawImage(image, this.position.x - size / 2, this.position.y - size / 2, size, size);

};

return Particle;

})();

/\*

\* ParticlePool class

\*/

var ParticlePool = (function() {

var particles,

firstActive = 0,

firstFree = 0,

duration = settings.particles.duration;

function ParticlePool(length) {

// create and populate particle pool

particles = new Array(length);

for (var i = 0; i < particles.length; i++)

particles[i] = new Particle();

}

ParticlePool.prototype.add = function(x, y, dx, dy) {

particles[firstFree].initialize(x, y, dx, dy);

// handle circular queue

firstFree++;

if (firstFree == particles.length) firstFree = 0;

if (firstActive == firstFree ) firstActive++;

if (firstActive == particles.length) firstActive = 0;

};

ParticlePool.prototype.update = function(deltaTime) {

var i;

// update active particles

if (firstActive < firstFree) {

for (i = firstActive; i < firstFree; i++)

particles[i].update(deltaTime);

}

if (firstFree < firstActive) {

for (i = firstActive; i < particles.length; i++)

particles[i].update(deltaTime);

for (i = 0; i < firstFree; i++)

particles[i].update(deltaTime);

}

// remove inactive particles

while (particles[firstActive].age >= duration && firstActive != firstFree) {

firstActive++;

if (firstActive == particles.length) firstActive = 0;

}

};

ParticlePool.prototype.draw = function(context, image) {

// draw active particles

if (firstActive < firstFree) {

for (i = firstActive; i < firstFree; i++)

particles[i].draw(context, image);

}

if (firstFree < firstActive) {

for (i = firstActive; i < particles.length; i++)

particles[i].draw(context, image);

for (i = 0; i < firstFree; i++)

particles[i].draw(context, image);

}

};

return ParticlePool;

})();

/\*

\* Putting it all together

\*/

(function(canvas) {

var context = canvas.getContext('2d'),

particles = new ParticlePool(settings.particles.length),

particleRate = settings.particles.length / settings.particles.duration, // particles/sec

time;

// get point on heart with -PI <= t <= PI

function pointOnHeart(t) {

return new Point(

160 \* Math.pow(Math.sin(t), 3),

130 \* Math.cos(t) - 50 \* Math.cos(2 \* t) - 20 \* Math.cos(3 \* t) - 10 \* Math.cos(4 \* t) + 25

);

}

// creating the particle image using a dummy canvas

var image = (function() {

var canvas = document.createElement('canvas'),

context = canvas.getContext('2d');

canvas.width = settings.particles.size;

canvas.height = settings.particles.size;

// helper function to create the path

function to(t) {

var point = pointOnHeart(t);

point.x = settings.particles.size / 2 + point.x \* settings.particles.size / 350;

point.y = settings.particles.size / 2 - point.y \* settings.particles.size / 350;

return point;

}

// create the path

context.beginPath();

var t = -Math.PI;

var point = to(t);

context.moveTo(point.x, point.y);

while (t < Math.PI) {

t += 0.01; // baby steps!

point = to(t);

context.lineTo(point.x, point.y);

}

context.closePath();

// create the fill

context.fillStyle = '#ea80b0';

context.fill();

// create the image

var image = new Image();

image.src = canvas.toDataURL();

return image;

})();

// render that thing!

function render() {

// next animation frame

requestAnimationFrame(render);

// update time

var newTime = new Date().getTime() / 1000,

deltaTime = newTime - (time || newTime);

time = newTime;

// clear canvas

context.clearRect(0, 0, canvas.width, canvas.height);

// create new particles

var amount = particleRate \* deltaTime;

for (var i = 0; i < amount; i++) {

var pos = pointOnHeart(Math.PI - 2 \* Math.PI \* Math.random());

var dir = pos.clone().length(settings.particles.velocity);

particles.add(canvas.width / 2 + pos.x, canvas.height / 2 - pos.y, dir.x, -dir.y);

}

// update and draw particles

particles.update(deltaTime);

particles.draw(context, image);

}

// handle (re-)sizing of the canvas

function onResize() {

canvas.width = canvas.clientWidth;

canvas.height = canvas.clientHeight;

}

window.onresize = onResize;

// delay rendering bootstrap

setTimeout(function() {

onResize();

render();

}, 10);

})(document.getElementById('pinkboard'));

var colours=new Array('#f00', '#f06', '#f0f', '#f6f', '#f39', '#f9c'); // colours of the hearts

var minisize=10; // smallest size of hearts in pixels

var maxisize=20; // biggest size of hearts in pixels

var hearts=100; // maximum number of hearts on screen

var over\_or\_under="over"; // set to "over" for hearts to always be on top, or "under" to allow them to float behind other objects

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*JavaScript Love Heart Cursor\*

\* (c)2013+ mf2fm web-design \*

\* http://www.mf2fm.com/rv \*

\* DON'T EDIT BELOW THIS BOX \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

var x=ox=400;

var y=oy=300;

var swide=800;

var shigh=600;

var sleft=sdown=0;

var herz=new Array();

var herzx=new Array();

var herzy=new Array();

var herzs=new Array();

var kiss=false;

if (typeof('addRVLoadEvent')!='function') function addRVLoadEvent(funky) {

var oldonload=window.onload;

if (typeof(oldonload)!='function') window.onload=funky;

else window.onload=function() {

if (oldonload) oldonload();

funky();

}

}

addRVLoadEvent(mwah);

function mwah() { if (document.getElementById) {

var i, heart;

for (i=0; i<hearts; i++) {

heart=createDiv("auto", "auto");

heart.style.visibility="hidden";

heart.style.zIndex=(over\_or\_under=="over")?"1001":"0";

heart.style.color=colours[i%colours.length];

heart.style.pointerEvents="none";

if (navigator.appName=="Microsoft Internet Explorer") heart.style.filter="alpha(opacity=75)";

else heart.style.opacity=0.45;

heart.appendChild(document.createTextNode(String.fromCharCode(9829)));

document.body.appendChild(heart);

herz[i]=heart;

herzy[i]=false;

}

set\_scroll();

set\_width();

herzle();

}}

function herzle() {

var c;

if (Math.abs(x-ox)>1 || Math.abs(y-oy)>1) {

ox=x;

oy=y;

for (c=0; c<hearts; c++) if (herzy[c]===false) {

herz[c].firstChild.nodeValue=String.fromCharCode(9829);

herz[c].style.left=(herzx[c]=x-minisize/2)+"px";

herz[c].style.top=(herzy[c]=y-minisize)+"px";

herz[c].style.fontSize=minisize+"px";

herz[c].style.fontWeight='normal';

herz[c].style.visibility='visible';

herzs[c]=minisize;

break;

}

}

for (c=0; c<hearts; c++) if (herzy[c]!==false) blow\_me\_a\_kiss(c);

setTimeout("herzle()", 30);

}

document.onmousedown=pucker;

document.onmouseup=function(){clearTimeout(kiss);};

function pucker() {

ox=-1;

oy=-1;

kiss=setTimeout('pucker()', 100);

}

function blow\_me\_a\_kiss(i) {

herzy[i]-=herzs[i]/minisize+i%2;

herzx[i]+=(i%5-2)/5;

if (herzy[i]<sdown-herzs[i] || herzx[i]<sleft-herzs[i] || herzx[i]>sleft+swide-herzs[i]) {

herz[i].style.visibility="hidden";

herzy[i]=false;

}

else if (herzs[i]>minisize+1 && Math.random()<2.5/hearts) break\_my\_heart(i);

else {

if (Math.random()<maxisize/herzy[i] && herzs[i]<maxisize) herz[i].style.fontSize=(++herzs[i])+"px";

herz[i].style.top=herzy[i]+"px";

herz[i].style.left=herzx[i]+"px";

}

}

function break\_my\_heart(i) {

var t;

herz[i].firstChild.nodeValue=String.fromCharCode(9676);

herz[i].style.fontWeight='bold';

herzy[i]=false;

for (t=herzs[i]; t<=maxisize; t++) setTimeout('herz['+i+'].style.fontSize="'+t+'px"', 60\*(t-herzs[i]));

setTimeout('herz['+i+'].style.visibility="hidden";', 60\*(t-herzs[i]));

}

document.onmousemove=mouse;

function mouse(e) {

if (e) {

y=e.pageY;

x=e.pageX;

}

else {

set\_scroll();

y=event.y+sdown;

x=event.x+sleft;

}

}

window.onresize=set\_width;

function set\_width() {

var sw\_min=999999;

var sh\_min=999999;

if (document.documentElement && document.documentElement.clientWidth) {

if (document.documentElement.clientWidth>0) sw\_min=document.documentElement.clientWidth;

if (document.documentElement.clientHeight>0) sh\_min=document.documentElement.clientHeight;

}

if (typeof(self.innerWidth)=='number' && self.innerWidth) {

if (self.innerWidth>0 && self.innerWidth<sw\_min) sw\_min=self.innerWidth;

if (self.innerHeight>0 && self.innerHeight<sh\_min) sh\_min=self.innerHeight;

}

if (document.body.clientWidth) {

if (document.body.clientWidth>0 && document.body.clientWidth<sw\_min) sw\_min=document.body.clientWidth;

if (document.body.clientHeight>0 && document.body.clientHeight<sh\_min) sh\_min=document.body.clientHeight;

}

if (sw\_min==999999 || sh\_min==999999) {

sw\_min=800;

sh\_min=600;

}

swide=sw\_min;

shigh=sh\_min;

}

window.onscroll=set\_scroll;

function set\_scroll() {

if (typeof(self.pageYOffset)=='number') {

sdown=self.pageYOffset;

sleft=self.pageXOffset;

}

else if (document.body && (document.body.scrollTop || document.body.scrollLeft)) {

sdown=document.body.scrollTop;

sleft=document.body.scrollLeft;

}

else if (document.documentElement && (document.documentElement.scrollTop || document.documentElement.scrollLeft)) {

sleft=document.documentElement.scrollLeft;

sdown=document.documentElement.scrollTop;

}

else {

sdown=0;

sleft=0;

}

}

function createDiv(height, width) {

var div=document.createElement("div");

div.style.position="absolute";

div.style.height=height;

div.style.width=width;

div.style.overflow="hidden";

div.style.backgroundColor="transparent";

return (div);

}

// ]]>

</script>

</BODY>

</HTML>

## **Làm code trái tim HTML huyền bí**

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>

<HEAD>

<TITLE> Heart </TITLE>

<META NAME="Generator" CONTENT="EditPlus">

<META NAME="Author" CONTENT="">

<META NAME="Keywords" CONTENT="">

<META NAME="Description" CONTENT="">

<style>

html, body {

height: 100%;

padding: 0;

margin: 0;

background: black;

}

canvas {

position: absolute;

width: 100%;

height: 100%;

}

</style>

</HEAD>

<BODY>

<canvas id="pinkboard"></canvas>

<script>

/\*

\* Settings

\*/

var settings = {

particles: {

length: 500, // maximum amount of particles

duration: 2, // particle duration in sec

velocity: 100, // particle velocity in pixels/sec

effect: -0.75, // play with this for a nice effect

size: 30, // particle size in pixels

},

};

/\*

\* RequestAnimationFrame polyfill by Erik Möller

\*/

(function(){var b=0;var c=["ms","moz","webkit","o"];for(var a=0;a<c.length&&!window.requestAnimationFrame;++a){window.requestAnimationFrame=window[c[a]+"RequestAnimationFrame"];window.cancelAnimationFrame=window[c[a]+"CancelAnimationFrame"]||window[c[a]+"CancelRequestAnimationFrame"]}if(!window.requestAnimationFrame){window.requestAnimationFrame=function(h,e){var d=new Date().getTime();var f=Math.max(0,16-(d-b));var g=window.setTimeout(function(){h(d+f)},f);b=d+f;return g}}if(!window.cancelAnimationFrame){window.cancelAnimationFrame=function(d){clearTimeout(d)}}}());

/\*

\* Point class

\*/

var Point = (function() {

function Point(x, y) {

this.x = (typeof x !== 'undefined') ? x : 0;

this.y = (typeof y !== 'undefined') ? y : 0;

}

Point.prototype.clone = function() {

return new Point(this.x, this.y);

};

Point.prototype.length = function(length) {

if (typeof length == 'undefined')

return Math.sqrt(this.x \* this.x + this.y \* this.y);

this.normalize();

this.x \*= length;

this.y \*= length;

return this;

};

Point.prototype.normalize = function() {

var length = this.length();

this.x /= length;

this.y /= length;

return this;

};

return Point;

})();

/\*

\* Particle class

\*/

var Particle = (function() {

function Particle() {

this.position = new Point();

this.velocity = new Point();

this.acceleration = new Point();

this.age = 0;

}

Particle.prototype.initialize = function(x, y, dx, dy) {

this.position.x = x;

this.position.y = y;

this.velocity.x = dx;

this.velocity.y = dy;

this.acceleration.x = dx \* settings.particles.effect;

this.acceleration.y = dy \* settings.particles.effect;

this.age = 0;

};

Particle.prototype.update = function(deltaTime) {

this.position.x += this.velocity.x \* deltaTime;

this.position.y += this.velocity.y \* deltaTime;

this.velocity.x += this.acceleration.x \* deltaTime;

this.velocity.y += this.acceleration.y \* deltaTime;

this.age += deltaTime;

};

Particle.prototype.draw = function(context, image) {

function ease(t) {

return (--t) \* t \* t + 1;

}

var size = image.width \* ease(this.age / settings.particles.duration);

context.globalAlpha = 1 - this.age / settings.particles.duration;

context.drawImage(image, this.position.x - size / 2, this.position.y - size / 2, size, size);

};

return Particle;

})();

/\*

\* ParticlePool class

\*/

var ParticlePool = (function() {

var particles,

firstActive = 0,

firstFree = 0,

duration = settings.particles.duration;

function ParticlePool(length) {

// create and populate particle pool

particles = new Array(length);

for (var i = 0; i < particles.length; i++)

particles[i] = new Particle();

}

ParticlePool.prototype.add = function(x, y, dx, dy) {

particles[firstFree].initialize(x, y, dx, dy);

// handle circular queue

firstFree++;

if (firstFree == particles.length) firstFree = 0;

if (firstActive == firstFree ) firstActive++;

if (firstActive == particles.length) firstActive = 0;

};

ParticlePool.prototype.update = function(deltaTime) {

var i;

// update active particles

if (firstActive < firstFree) {

for (i = firstActive; i < firstFree; i++)

particles[i].update(deltaTime);

}

if (firstFree < firstActive) {

for (i = firstActive; i < particles.length; i++)

particles[i].update(deltaTime);

for (i = 0; i < firstFree; i++)

particles[i].update(deltaTime);

}

// remove inactive particles

while (particles[firstActive].age >= duration && firstActive != firstFree) {

firstActive++;

if (firstActive == particles.length) firstActive = 0;

}

};

ParticlePool.prototype.draw = function(context, image) {

// draw active particles

if (firstActive < firstFree) {

for (i = firstActive; i < firstFree; i++)

particles[i].draw(context, image);

}

if (firstFree < firstActive) {

for (i = firstActive; i < particles.length; i++)

particles[i].draw(context, image);

for (i = 0; i < firstFree; i++)

particles[i].draw(context, image);

}

};

return ParticlePool;

})();

/\*

\* Putting it all together

\*/

(function(canvas) {

var context = canvas.getContext('2d'),

particles = new ParticlePool(settings.particles.length),

particleRate = settings.particles.length / settings.particles.duration, // particles/sec

time;

// get point on heart with -PI <= t <= PI

function pointOnHeart(t) {

return new Point(

160 \* Math.pow(Math.sin(t), 3),

130 \* Math.cos(t) - 50 \* Math.cos(2 \* t) - 20 \* Math.cos(3 \* t) - 10 \* Math.cos(4 \* t) + 25

);

}

// creating the particle image using a dummy canvas

var image = (function() {

var canvas = document.createElement('canvas'),

context = canvas.getContext('2d');

canvas.width = settings.particles.size;

canvas.height = settings.particles.size;

// helper function to create the path

function to(t) {

var point = pointOnHeart(t);

point.x = settings.particles.size / 2 + point.x \* settings.particles.size / 350;

point.y = settings.particles.size / 2 - point.y \* settings.particles.size / 350;

return point;

}

// create the path

context.beginPath();

var t = -Math.PI;

var point = to(t);

context.moveTo(point.x, point.y);

while (t < Math.PI) {

t += 0.01; // baby steps!

point = to(t);

context.lineTo(point.x, point.y);

}

context.closePath();

// create the fill

context.fillStyle = '#ea80b0';

context.fill();

// create the image

var image = new Image();

image.src = canvas.toDataURL();

return image;

})();

// render that thing!

function render() {

// next animation frame

requestAnimationFrame(render);

// update time

var newTime = new Date().getTime() / 1000,

deltaTime = newTime - (time || newTime);

time = newTime;

// clear canvas

context.clearRect(0, 0, canvas.width, canvas.height);

// create new particles

var amount = particleRate \* deltaTime;

for (var i = 0; i < amount; i++) {

var pos = pointOnHeart(Math.PI - 2 \* Math.PI \* Math.random());

var dir = pos.clone().length(settings.particles.velocity);

particles.add(canvas.width / 2 + pos.x, canvas.height / 2 - pos.y, dir.x, -dir.y);

}

// update and draw particles

particles.update(deltaTime);

particles.draw(context, image);

}

// handle (re-)sizing of the canvas

function onResize() {

canvas.width = canvas.clientWidth;

canvas.height = canvas.clientHeight;

}

window.onresize = onResize;

// delay rendering bootstrap

setTimeout(function() {

onResize();

render();

}, 10);

})(document.getElementById('pinkboard'));

var colours=new Array('#f00', '#f06', '#f0f', '#f6f', '#f39', '#f9c'); // colours of the hearts

var minisize=10; // smallest size of hearts in pixels

var maxisize=20; // biggest size of hearts in pixels

var hearts=100; // maximum number of hearts on screen

var over\_or\_under="over"; // set to "over" for hearts to always be on top, or "under" to allow them to float behind other objects

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*JavaScript Love Heart Cursor\*

\* (c)2013+ mf2fm web-design \*

\* http://www.mf2fm.com/rv \*

\* DON'T EDIT BELOW THIS BOX \*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

var x=ox=400;

var y=oy=300;

var swide=800;

var shigh=600;

var sleft=sdown=0;

var herz=new Array();

var herzx=new Array();

var herzy=new Array();

var herzs=new Array();

var kiss=false;

if (typeof('addRVLoadEvent')!='function') function addRVLoadEvent(funky) {

var oldonload=window.onload;

if (typeof(oldonload)!='function') window.onload=funky;

else window.onload=function() {

if (oldonload) oldonload();

funky();

}

}

addRVLoadEvent(mwah);

function mwah() { if (document.getElementById) {

var i, heart;

for (i=0; i<hearts; i++) {

heart=createDiv("auto", "auto");

heart.style.visibility="hidden";

heart.style.zIndex=(over\_or\_under=="over")?"1001":"0";

heart.style.color=colours[i%colours.length];

heart.style.pointerEvents="none";

if (navigator.appName=="Microsoft Internet Explorer") heart.style.filter="alpha(opacity=75)";

else heart.style.opacity=0.45;

heart.appendChild(document.createTextNode(String.fromCharCode(9829)));

document.body.appendChild(heart);

herz[i]=heart;

herzy[i]=false;

}

set\_scroll();

set\_width();

herzle();

}}

function herzle() {

var c;

if (Math.abs(x-ox)>1 || Math.abs(y-oy)>1) {

ox=x;

oy=y;

for (c=0; c<hearts; c++) if (herzy[c]===false) {

herz[c].firstChild.nodeValue=String.fromCharCode(9829);

herz[c].style.left=(herzx[c]=x-minisize/2)+"px";

herz[c].style.top=(herzy[c]=y-minisize)+"px";

herz[c].style.fontSize=minisize+"px";

herz[c].style.fontWeight='normal';

herz[c].style.visibility='visible';

herzs[c]=minisize;

break;

}

}

for (c=0; c<hearts; c++) if (herzy[c]!==false) blow\_me\_a\_kiss(c);

setTimeout("herzle()", 30);

}

document.onmousedown=pucker;

document.onmouseup=function(){clearTimeout(kiss);};

function pucker() {

ox=-1;

oy=-1;

kiss=setTimeout('pucker()', 100);

}

function blow\_me\_a\_kiss(i) {

herzy[i]-=herzs[i]/minisize+i%2;

herzx[i]+=(i%5-2)/5;

if (herzy[i]<sdown-herzs[i] || herzx[i]<sleft-herzs[i] || herzx[i]>sleft+swide-herzs[i]) {

herz[i].style.visibility="hidden";

herzy[i]=false;

}

else if (herzs[i]>minisize+1 && Math.random()<2.5/hearts) break\_my\_heart(i);

else {

if (Math.random()<maxisize/herzy[i] && herzs[i]<maxisize) herz[i].style.fontSize=(++herzs[i])+"px";

herz[i].style.top=herzy[i]+"px";

herz[i].style.left=herzx[i]+"px";

}

}

function break\_my\_heart(i) {

var t;

herz[i].firstChild.nodeValue=String.fromCharCode(9676);

herz[i].style.fontWeight='bold';

herzy[i]=false;

for (t=herzs[i]; t<=maxisize; t++) setTimeout('herz['+i+'].style.fontSize="'+t+'px"', 60\*(t-herzs[i]));

setTimeout('herz['+i+'].style.visibility="hidden";', 60\*(t-herzs[i]));

}

document.onmousemove=mouse;

function mouse(e) {

if (e) {

y=e.pageY;

x=e.pageX;

}

else {

set\_scroll();

y=event.y+sdown;

x=event.x+sleft;

}

}

window.onresize=set\_width;

function set\_width() {

var sw\_min=999999;

var sh\_min=999999;

if (document.documentElement && document.documentElement.clientWidth) {

if (document.documentElement.clientWidth>0) sw\_min=document.documentElement.clientWidth;

if (document.documentElement.clientHeight>0) sh\_min=document.documentElement.clientHeight;

}

if (typeof(self.innerWidth)=='number' && self.innerWidth) {

if (self.innerWidth>0 && self.innerWidth<sw\_min) sw\_min=self.innerWidth;

if (self.innerHeight>0 && self.innerHeight<sh\_min) sh\_min=self.innerHeight;

}

if (document.body.clientWidth) {

if (document.body.clientWidth>0 && document.body.clientWidth<sw\_min) sw\_min=document.body.clientWidth;

if (document.body.clientHeight>0 && document.body.clientHeight<sh\_min) sh\_min=document.body.clientHeight;

}

if (sw\_min==999999 || sh\_min==999999) {

sw\_min=800;

sh\_min=600;

}

swide=sw\_min;

shigh=sh\_min;

}

window.onscroll=set\_scroll;

function set\_scroll() {

if (typeof(self.pageYOffset)=='number') {

sdown=self.pageYOffset;

sleft=self.pageXOffset;

}

else if (document.body && (document.body.scrollTop || document.body.scrollLeft)) {

sdown=document.body.scrollTop;

sleft=document.body.scrollLeft;

}

else if (document.documentElement && (document.documentElement.scrollTop || document.documentElement.scrollLeft)) {

sleft=document.documentElement.scrollLeft;

sdown=document.documentElement.scrollTop;

}

else {

sdown=0;

sleft=0;

}

}

function createDiv(height, width) {

var div=document.createElement("div");

div.style.position="absolute";

div.style.height=height;

div.style.width=width;

div.style.overflow="hidden";

div.style.backgroundColor="transparent";

return (div);

}

// ]]>

</script>

</BODY>

</HTML>

## **Làm code trái tim HTML đập chèn chữ**

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>

<HEAD>

<TITLE> Heart </TITLE>

<META NAME="Generator" CONTENT="EditPlus">

<META NAME="Author" CONTENT="">

<META NAME="Keywords" CONTENT="">

<META NAME="Description" CONTENT="">

<style>

html, body {

height: 100%;

padding: 0;

margin: 0;

background: #000;

display: flex;

justify-content: center;

align-items: center;

}

.box {

width: 100%;

position: absolute;

top: 50%;

left: 50%;

transform: translate(-50%, -50%);

display: flex;

flex-direction: column;

}

canvas {

position: absolute;

width: 100%;

height: 100%;

}

#pinkboard {

position: relative;

margin: auto;

height: 500px;

width: 500px;

animation: animate 1.3s infinite;

}

#pinkboard:before, #pinkboard:after {

content: '';

position: absolute;

background: #FF5CA4;

width: 100px;

height: 160px;

border-top-left-radius: 50px;

border-top-right-radius: 50px;

}

#pinkboard:before {

left: 100px;

transform: rotate(-45deg);

transform-origin: 0 100%;

box-shadow: 0 14px 28px rgba(0,0,0,0.25),

0 10px 10px rgba(0,0,0,0.22);

}

#pinkboard:after {

left: 0;

transform: rotate(45deg);

transform-origin: 100% 100%;

}

@keyframes animate {

0% {

transform: scale(1);

}

30% {

transform: scale(.8);

}

60% {

transform: scale(1.2);

}

100% {

transform: scale(1);

}

}

</style>

</HEAD>

<BODY>

<div class="box">

<canvas id="pinkboard"></canvas>

</div>

<script>

/\*

\* Settings

\*/

var settings = {

particles: {

length: 2000, // maximum amount of particles

duration: 2, // particle duration in sec

velocity: 100, // particle velocity in pixels/sec

effect: -1.3, // play with this for a nice effect

size: 13, // particle size in pixels

},

};

/\*

\* RequestAnimationFrame polyfill by Erik Möller

\*/

(function(){var b=0;var c=["ms","moz","webkit","o"];for(var a=0;a<c.length&&!window.requestAnimationFrame;++a){window.requestAnimationFrame=window[c[a]+"RequestAnimationFrame"];window.cancelAnimationFrame=window[c[a]+"CancelAnimationFrame"]||window[c[a]+"CancelRequestAnimationFrame"]}if(!window.requestAnimationFrame){window.requestAnimationFrame=function(h,e){var d=new Date().getTime();var f=Math.max(0,16-(d-b));var g=window.setTimeout(function(){h(d+f)},f);b=d+f;return g}}if(!window.cancelAnimationFrame){window.cancelAnimationFrame=function(d){clearTimeout(d)}}}());

/\*

\* Point class

\*/

var Point = (function() {

function Point(x, y) {

this.x = (typeof x !== 'undefined') ? x : 0;

this.y = (typeof y !== 'undefined') ? y : 0;

}

Point.prototype.clone = function() {

return new Point(this.x, this.y);

};

Point.prototype.length = function(length) {

if (typeof length == 'undefined')

return Math.sqrt(this.x \* this.x + this.y \* this.y);

this.normalize();

this.x \*= length;

this.y \*= length;

return this;

};

Point.prototype.normalize = function() {

var length = this.length();

this.x /= length;

this.y /= length;

return this;

};

return Point;

})();

/\*

\* Particle class

\*/

var Particle = (function() {

function Particle() {

this.position = new Point();

this.velocity = new Point();

this.acceleration = new Point();

this.age = 0;

}

Particle.prototype.initialize = function(x, y, dx, dy) {

this.position.x = x;

this.position.y = y;

this.velocity.x = dx;

this.velocity.y = dy;

this.acceleration.x = dx \* settings.particles.effect;

this.acceleration.y = dy \* settings.particles.effect;

this.age = 0;

};

Particle.prototype.update = function(deltaTime) {

this.position.x += this.velocity.x \* deltaTime;

this.position.y += this.velocity.y \* deltaTime;

this.velocity.x += this.acceleration.x \* deltaTime;

this.velocity.y += this.acceleration.y \* deltaTime;

this.age += deltaTime;

};

Particle.prototype.draw = function(context, image) {

function ease(t) {

return (--t) \* t \* t + 1;

}

var size = image.width \* ease(this.age / settings.particles.duration);

context.globalAlpha = 1 - this.age / settings.particles.duration;

context.drawImage(image, this.position.x - size / 2, this.position.y - size / 2, size, size);

};

return Particle;

})();

/\*

\* ParticlePool class

\*/

var ParticlePool = (function() {

var particles,

firstActive = 0,

firstFree = 0,

duration = settings.particles.duration;

function ParticlePool(length) {

// create and populate particle pool

particles = new Array(length);

for (var i = 0; i < particles.length; i++)

particles[i] = new Particle();

}

ParticlePool.prototype.add = function(x, y, dx, dy) {

particles[firstFree].initialize(x, y, dx, dy);

// handle circular queue

firstFree++;

if (firstFree == particles.length) firstFree = 0;

if (firstActive == firstFree ) firstActive++;

if (firstActive == particles.length) firstActive = 0;

};

ParticlePool.prototype.update = function(deltaTime) {

var i;

// update active particles

if (firstActive < firstFree) {

for (i = firstActive; i < firstFree; i++)

particles[i].update(deltaTime);

}

if (firstFree < firstActive) {

for (i = firstActive; i < particles.length; i++)

particles[i].update(deltaTime);

for (i = 0; i < firstFree; i++)

particles[i].update(deltaTime);

}

// remove inactive particles

while (particles[firstActive].age >= duration && firstActive != firstFree) {

firstActive++;

if (firstActive == particles.length) firstActive = 0;

}

};

ParticlePool.prototype.draw = function(context, image) {

// draw active particles

if (firstActive < firstFree) {

for (i = firstActive; i < firstFree; i++)

particles[i].draw(context, image);

}

if (firstFree < firstActive) {

for (i = firstActive; i < particles.length; i++)

particles[i].draw(context, image);

for (i = 0; i < firstFree; i++)

particles[i].draw(context, image);

}

};

return ParticlePool;

})();

/\*

\* Putting it all together

\*/

(function(canvas) {

var context = canvas.getContext('2d'),

particles = new ParticlePool(settings.particles.length),

particleRate = settings.particles.length / settings.particles.duration, // particles/sec

time;

// get point on heart with -PI <= t <= PI

function pointOnHeart(t) {

return new Point(

160 \* Math.pow(Math.sin(t), 3),

130 \* Math.cos(t) - 50 \* Math.cos(2 \* t) - 20 \* Math.cos(3 \* t) - 10 \* Math.cos(4 \* t) + 25

);

}

// creating the particle image using a dummy canvas

var image = (function() {

var canvas = document.createElement('canvas'),

context = canvas.getContext('2d');

canvas.width = settings.particles.size;

canvas.height = settings.particles.size;

// helper function to create the path

function to(t) {

var point = pointOnHeart(t);

point.x = settings.particles.size / 2 + point.x \* settings.particles.size / 350;

point.y = settings.particles.size / 2 - point.y \* settings.particles.size / 350;

return point;

}

// create the path

context.beginPath();

var t = -Math.PI;

var point = to(t);

context.moveTo(point.x, point.y);

while (t < Math.PI) {

t += 0.01; // baby steps!

point = to(t);

context.lineTo(point.x, point.y);

}

context.closePath();

// create the fill

context.fillStyle = '#FF5CA4';

context.fill();

// create the image

var image = new Image();

image.src = canvas.toDataURL();

return image;

})();

// render that thing!

function render() {

// next animation frame

requestAnimationFrame(render);

// update time

var newTime = new Date().getTime() / 1000,

deltaTime = newTime - (time || newTime);

time = newTime;

// clear canvas

context.clearRect(0, 0, canvas.width, canvas.height);

// create new particles

var amount = particleRate \* deltaTime;

for (var i = 0; i < amount; i++) {

var pos = pointOnHeart(Math.PI - 2 \* Math.PI \* Math.random());

var dir = pos.clone().length(settings.particles.velocity);

particles.add(canvas.width / 2 + pos.x, canvas.height / 2 - pos.y, dir.x, -dir.y);

}

// update and draw particles

particles.update(deltaTime);

particles.draw(context, image);

}

// handle (re-)sizing of the canvas

function onResize() {

canvas.width = canvas.clientWidth;

canvas.height = canvas.clientHeight;

}

window.onresize = onResize;

// delay rendering bootstrap

setTimeout(function() {

onResize();

render();

}, 10);

})(document.getElementById('pinkboard'));

</script>

<div class="center-text",

style="background-color:rgb(0, 0, 0);

width: 100%;

color: rgb(225, 12, 168);

height:100%;

font-size: 31px;

font-style: italic;

display: flex;

align-items: center;

justify-content: center;

margin-bottom: 5px;

text-align: center;">I Love You</div>

</BODY>

</HTML>

## **Làm code trái tim thật có nhịp đập**

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">

<HTML>

<HEAD>

<TITLE> MINH IT </TITLE>

<META NAME="Generator" CONTENT="EditPlus">

<META NAME="Author" CONTENT="">

<META NAME="Keywords" CONTENT="">

<META NAME="Description" CONTENT="">

<link rel="stylesheet" href="style.css">

<style>

html, body {

height: 100%;

padding: 0;

margin: 0;

background: rgba(0, 0, 0, 0.851);

}

canvas {

position: absolute;

width: 100%;

height: 100%;

}

</style>

</HEAD>

<BODY>

<div class="box">

<canvas id="pinkboard"></canvas>

</div>

<script>

var settings = {

particles: {

length: 10000, // maximum amount of particles

duration: 4, // particle duration in sec

velocity: 80, // particle velocity in pixels/sec

effect: -1.3, // play with this for a nice effect

size: 8, // particle size in pixels

},

};

/\*

\*/

(function(){var b=0;var c=["ms","moz","webkit","o"];for(var a=0;a<c.length&&!window.requestAnimationFrame;++a){window.requestAnimationFrame=window[c[a]+"RequestAnimationFrame"];window.cancelAnimationFrame=window[c[a]+"CancelAnimationFrame"]||window[c[a]+"CancelRequestAnimationFrame"]}if(!window.requestAnimationFrame){window.requestAnimationFrame=function(h,e){var d=new Date().getTime();var f=Math.max(0,16-(d-b));var g=window.setTimeout(function(){h(d+f)},f);b=d+f;return g}}if(!window.cancelAnimationFrame){window.cancelAnimationFrame=function(d){clearTimeout(d)}}}());

/\*

\* Point class

\*/

var Point = (function() {

function Point(x, y) {

this.x = (typeof x !== 'undefined') ? x : 0;

this.y = (typeof y !== 'undefined') ? y : 0;

}

Point.prototype.clone = function() {

return new Point(this.x, this.y);

};

Point.prototype.length = function(length) {

if (typeof length == 'undefined')

return Math.sqrt(this.x \* this.x + this.y \* this.y);

this.normalize();

this.x \*= length;

this.y \*= length;

return this;

};

Point.prototype.normalize = function() {

var length = this.length();

this.x /= length;

this.y /= length;

return this;

};

return Point;

})();

/\*

\* Particle class

\*/

var Particle = (function() {

function Particle() {

this.position = new Point();

this.velocity = new Point();

this.acceleration = new Point();

this.age = 0;

}

Particle.prototype.initialize = function(x, y, dx, dy) {

this.position.x = x;

this.position.y = y;

this.velocity.x = dx;

this.velocity.y = dy;

this.acceleration.x = dx \* settings.particles.effect;

this.acceleration.y = dy \* settings.particles.effect;

this.age = 0;

};

Particle.prototype.update = function(deltaTime) {

this.position.x += this.velocity.x \* deltaTime;

this.position.y += this.velocity.y \* deltaTime;

this.velocity.x += this.acceleration.x \* deltaTime;

this.velocity.y += this.acceleration.y \* deltaTime;

this.age += deltaTime;

};

Particle.prototype.draw = function(context, image) {

function ease(t) {

return (--t) \* t \* t + 1;

}

var size = image.width \* ease(this.age / settings.particles.duration);

context.globalAlpha = 1 - this.age / settings.particles.duration;

context.drawImage(image, this.position.x - size / 2, this.position.y - size / 2, size, size);

};

return Particle;

})();

/\*

\* ParticlePool class

\*/

var ParticlePool = (function() {

var particles,

firstActive = 0,

firstFree = 0,

duration = settings.particles.duration;

function ParticlePool(length) {

// create and populate particle pool

particles = new Array(length);

for (var i = 0; i < particles.length; i++)

particles[i] = new Particle();

}

ParticlePool.prototype.add = function(x, y, dx, dy) {

particles[firstFree].initialize(x, y, dx, dy);

// handle circular queue

firstFree++;

if (firstFree == particles.length) firstFree = 0;

if (firstActive == firstFree ) firstActive++;

if (firstActive == particles.length) firstActive = 0;

};

ParticlePool.prototype.update = function(deltaTime) {

var i;

// update active particles

if (firstActive < firstFree) {

for (i = firstActive; i < firstFree; i++)

particles[i].update(deltaTime);

}

if (firstFree < firstActive) {

for (i = firstActive; i < particles.length; i++)

particles[i].update(deltaTime);

for (i = 0; i < firstFree; i++)

particles[i].update(deltaTime);

}

// remove inactive particles

while (particles[firstActive].age >= duration && firstActive != firstFree) {

firstActive++;

if (firstActive == particles.length) firstActive = 0;

}

};

ParticlePool.prototype.draw = function(context, image) {

// draw active particles

if (firstActive < firstFree) {

for (i = firstActive; i < firstFree; i++)

particles[i].draw(context, image);

}

if (firstFree < firstActive) {

for (i = firstActive; i < particles.length; i++)

particles[i].draw(context, image);

for (i = 0; i < firstFree; i++)

particles[i].draw(context, image);

}

};

return ParticlePool;

})();

/\*

\* Putting it all together

\*/

(function(canvas) {

var context = canvas.getContext('2d'),

particles = new ParticlePool(settings.particles.length),

particleRate = settings.particles.length / settings.particles.duration, // particles/sec

time;

// get point on heart with -PI <= t <= PI

function pointOnHeart(t) {

return new Point(

160 \* Math.pow(Math.sin(t), 3),

130 \* Math.cos(t) - 50 \* Math.cos(2 \* t) - 20 \* Math.cos(3 \* t) - 10 \* Math.cos(4 \* t) + 25

);

}

// creating the particle image using a dummy canvas

var image = (function() {

var canvas = document.createElement('canvas'),

context = canvas.getContext('2d');

canvas.width = settings.particles.size;

canvas.height = settings.particles.size;

// helper function to create the path

function to(t) {

var point = pointOnHeart(t);

point.x = settings.particles.size / 2 + point.x \* settings.particles.size / 350;

point.y = settings.particles.size / 2 - point.y \* settings.particles.size / 350;

return point;

}

// create the path

context.beginPath();

var t = -Math.PI;

var point = to(t);

context.moveTo(point.x, point.y);

while (t < Math.PI) {

t += 0.01; // baby steps!

point = to(t);

context.lineTo(point.x, point.y);

}

context.closePath();

// create the fill

context.fillStyle = '#f50b02';

context.fill();

// create the image

var image = new Image();

image.src = canvas.toDataURL();

return image;

})();

// render that thing!

function render() {

// next animation frame

requestAnimationFrame(render);

// update time

var newTime = new Date().getTime() / 1000,

deltaTime = newTime - (time || newTime);

time = newTime;

// clear canvas

context.clearRect(0, 0, canvas.width, canvas.height);

// create new particles

var amount = particleRate \* deltaTime;

for (var i = 0; i < amount; i++) {

var pos = pointOnHeart(Math.PI - 2 \* Math.PI \* Math.random());

var dir = pos.clone().length(settings.particles.velocity);

particles.add(canvas.width / 2 + pos.x, canvas.height / 2 - pos.y, dir.x, -dir.y);

}

// update and draw particles

particles.update(deltaTime);

particles.draw(context, image);

}

// handle (re-)sizing of the canvas

function onResize() {

canvas.width = canvas.clientWidth;

canvas.height = canvas.clientHeight;

}

window.onresize = onResize;

// delay rendering bootstrap

setTimeout(function() {

onResize();

render();

}, 10);

})(document.getElementById('pinkboard'));

</script>

</BODY>

</HTML>

## **Làm code trái tim kim cương với hiệu ứng màu sắc rực rỡ**

<!DOCTYPE html>

<html lang="en"><head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>I Love You</title>

<style>

\*{

margin: 0;

padding: 0;

box-sizing: border-box;

}

html,

body {

overflow: hidden;

background-color: #000 !important;

}

body {

-webkit-font-soothing: antialiased;

}

.webgl {

position: fixed;

width: 100vw;

height: 100vw;

top: 0;

left: 0;

outline: none;

}

h1{

position: absolute;

top: 10vh;

left: 2.5rem;

right: 1rem;

text-align: center;

font-size: max(1rem, 3vh);

}

button {

position: absolute;

left: 0;

top: 0;

bottom: 0;

height: 12vh;

width: 12vh;

transform: translateY(2vh);

right: 0;

margin: auto;

-webkit-appearance: none;

background: transparent;

color: inherit;

border: none;

cursor: pointer;

}

svg{

width: 3.5vh;

}

</style>

<script>

window.console = window.console || function(t) {};

</script>

<script>

if (document.location.search.match(/type=embed/gi)) {

window.parent.postMessage("resize", "\*");

}

</script>

</head>

<body translate="no">

<canvas class="webgl" data-engine="three.js r135" width="682" height="157" style="width: 682px; height: 157px;"></canvas>

<button id="play-music" type="button" aria-label="Play music" style="opacity: 1;"><svg fill="currentColor" viewBox="0 0 512 512" width="100" title="music">

<!-- <path d="M470.38 1.51L150.41 96A32 32 0 0 0 128 126.51v261.41A139 139 0 0 0 96 384c-53 0-96 28.66-96 64s43 64 96 64 96-28.66 96-64V214.32l256-75v184.61a138.4 138.4 0 0 0-32-3.93c-53 0-96 28.66-96 64s43 64 96 64 96-28.65 96-64V32a32 32 0 0 0-41.62-30.49z" /> -->

</svg></button>

<script type="x-shader/x-vertex" id="vertexShader">

#define M\_PI 3.1415926535897932384626433832795

uniform float uTime;

uniform float uSize;

attribute float aScale;

attribute vec3 aColor;

attribute float random;

attribute float random1;

attribute float aSpeed;

varying vec3 vColor;

varying vec2 vUv;

void main() {

float sign = 2.0\* (step(random, 0.5) -.5);

float t = sign\*mod(-uTime \* aSpeed\* 0.005 + 10.0\*aSpeed\*aSpeed, M\_PI);

float a = pow(t, 2.0) \* pow((t - sign \* M\_PI), 2.0);

float radius = 0.14;

vec3 myOffset =

vec3(t, 1.0, 0.0);

myOffset = vec3(radius \*16.0 \* pow(sin(t), 2.0) \* sin(t), radius \* (13.0 \* cos(t) - 5.0 \* cos(2.0 \* t) - 2.0 \* cos(3.0 \* t) - cos(4.0 \* t)), .15\*(a\*(random1 - .5))\*sin(abs(10.0\*(sin(.2\*uTime + .2\*random)))\*t));

vec3 displacedPosition = myOffset;

vec4 modelPosition = modelMatrix \* vec4(displacedPosition.xyz, 1.0);

vec4 viewPosition = viewMatrix \* modelPosition;

viewPosition.xyz += position \* aScale \* uSize \* pow(a, .5) \* .5;

gl\_Position = projectionMatrix \* viewPosition;

vColor = aColor;

vUv = uv;

}

</script>

<script type="x-shader/x-fragment" id="fragmentShader">

varying vec3 vColor;

varying vec2 vUv;

void main() {

vec2 uv = vUv;

vec3 color = vColor;

float strength = distance(uv, vec2(0.5));

strength \*= 2.0;

strength = 1.0 - strength;

gl\_FragColor = vec4(strength \* color, 1.0);

}

</script>

<script type="x-shader/x-vertex" id="vertexShader1">

#define M\_PI 3.1415926535897932384626433832795

uniform float uTime;

uniform float uSize;

attribute float aScale;

attribute vec3 aColor;

attribute float phi;

attribute float random;

attribute float random1;

varying vec3 vColor;

varying vec2 vUv;

void main() {

float t = 0.01 \* uTime + 12.0;

float angle = phi;

t = mod((-uTime + 100.0) \* 0.06\* random1 + random \*2.0 \* M\_PI , 2.0 \* M\_PI);

vec3 myOffset = vec3(5.85\*cos(angle \* (t )), 2.0\*(t - M\_PI), 3.0\*sin(angle \* (t )/t));

vec4 modelPosition = modelMatrix \* vec4(myOffset, 1.0);

vec4 viewPosition = viewMatrix \* modelPosition;

viewPosition.xyz += position \* aScale \* uSize;

gl\_Position = projectionMatrix \* viewPosition;

vColor = aColor;

vUv = uv;

}

</script>

<script type="x-shader/x-fragment" id="fragmentShader1">

uniform sampler2D uTex;

varying vec3 vColor;

varying vec2 vUv;

void main() {

vec2 uv = vUv;

vec3 color = vColor;

float strength = distance(uv, vec2(0.5, .65));

strength \*= 2.0;

strength = 1.0 - strength;

vec3 texture = texture2D(uTex, uv).rgb;

gl\_FragColor = vec4(texture \* color \* (strength + .3), 1.);

}

</script>

<script src="https://cpwebassets.codepen.io/assets/common/stopExecutionOnTimeout-2c7831bb44f98c1391d6a4ffda0e1fd302503391ca806e7fcc7b9b87197aec26.js"></script>

<script id="rendered-js" type="module">

/\* Poly Heart model by Quaternius [CC0] (https://creativecommons.org/publicdomain/zero/1.0/) via Poly Pizza (https://poly.pizza/m/1yCRUwFnwX)

\*/

import \* as THREE from "https://cdn.skypack.dev/three@0.135.0";

import { gsap } from "https://cdn.skypack.dev/gsap@3.8.0";

import { GLTFLoader } from "https://cdn.skypack.dev/three@0.135.0/examples/jsm/loaders/GLTFLoader";

class World {

constructor({

canvas,

width,

height,

cameraPosition,

fieldOfView = 75,

nearPlane = 0.1,

farPlane = 100 })

{

this.parameters = {

count: 1500,

max: 12.5 \* Math.PI,

a: 2,

c: 4.5 };

this.textureLoader = new THREE.TextureLoader();

this.scene = new THREE.Scene();

this.scene.background = new THREE.Color(0x16000a);

this.clock = new THREE.Clock();

this.data = 0;

this.time = { current: 0, t0: 0, t1: 0, t: 0, frequency: 0.0005 };

this.angle = { x: 0, z: 0 };

this.width = width || window.innerWidth;

this.height = height || window.innerHeight;

this.aspectRatio = this.width / this.height;

this.fieldOfView = fieldOfView;

this.camera = new THREE.PerspectiveCamera(

fieldOfView,

this.aspectRatio,

nearPlane,

farPlane);

this.camera.position.set(

cameraPosition.x,

cameraPosition.y,

cameraPosition.z);

this.scene.add(this.camera);

this.renderer = new THREE.WebGLRenderer({

canvas,

antialias: true });

this.pixelRatio = Math.min(window.devicePixelRatio, 2);

this.renderer.setPixelRatio(this.pixelRatio);

this.renderer.setSize(this.width, this.height);

this.timer = 0;

this.addToScene();

this.addButton();

this.render();

this.listenToResize();

this.listenToMouseMove();

}

start() {}

render() {

this.renderer.render(this.scene, this.camera);

this.composer && this.composer.render();

}

loop() {

this.time.elapsed = this.clock.getElapsedTime();

this.time.delta = Math.min(

60,

(this.time.current - this.time.elapsed) \* 1000);

if (this.analyser && this.isRunning) {

this.time.t = this.time.elapsed - this.time.t0 + this.time.t1;

this.data = this.analyser.getAverageFrequency();

this.data \*= this.data / 2000;

this.angle.x += this.time.delta \* 0.001 \* 0.63;

this.angle.z += this.time.delta \* 0.001 \* 0.39;

const justFinished = this.isRunning && !this.sound.isPlaying;

if (justFinished) {

this.time.t1 = this.time.t;

this.audioBtn.disabled = false;

this.isRunning = false;

const tl = gsap.timeline();

this.angle.x = 0;

this.angle.z = 0;

tl.to(this.camera.position, {

x: 0,

z: 4.5,

duration: 4,

ease: "expo.in" });

tl.to(this.audioBtn, {

opacity: () => 1,

duration: 1,

ease: "power1.out" });

} else {

this.camera.position.x = Math.sin(this.angle.x) \* this.parameters.a;

this.camera.position.z = Math.min(

Math.max(Math.cos(this.angle.z) \* this.parameters.c, 1.75),

6.5);

}

}

this.camera.lookAt(this.scene.position);

if (this.heartMaterial) {

this.heartMaterial.uniforms.uTime.value +=

this.time.delta \* this.time.frequency \* (1 + this.data \* 0.2);

}

if (this.model) {

this.model.rotation.y -= 0.0005 \* this.time.delta \* (1 + this.data);

}

if (this.snowMaterial) {

this.snowMaterial.uniforms.uTime.value +=

this.time.delta \* 0.0004 \* (1 + this.data);

}

this.render();

this.time.current = this.time.elapsed;

requestAnimationFrame(this.loop.bind(this));

}

listenToResize() {

window.addEventListener("resize", () => {

// Update sizes

this.width = window.innerWidth;

this.height = window.innerHeight;

// Update camera

this.camera.aspect = this.width / this.height;

this.camera.updateProjectionMatrix();

this.renderer.setSize(this.width, this.height);

});

}

listenToMouseMove() {

window.addEventListener("mousemove", e => {

const x = e.clientX;

const y = e.clientY;

gsap.to(this.camera.position, {

x: gsap.utils.mapRange(0, window.innerWidth, 0.2, -0.2, x),

y: gsap.utils.mapRange(0, window.innerHeight, 0.2, -0.2, -y) });

});

}

addHeart() {

this.heartMaterial = new THREE.ShaderMaterial({

fragmentShader: document.getElementById("fragmentShader").textContent,

vertexShader: document.getElementById("vertexShader").textContent,

uniforms: {

uTime: { value: 0 },

uSize: { value: 0.2 },

uTex: {

value: new THREE.TextureLoader().load(

"https://assets.codepen.io/74321/heart.png") } },

depthWrite: false,

blending: THREE.AdditiveBlending,

transparent: true });

const count = this.parameters.count; //2000

const scales = new Float32Array(count \* 1);

const colors = new Float32Array(count \* 3);

const speeds = new Float32Array(count);

const randoms = new Float32Array(count);

const randoms1 = new Float32Array(count);

const colorChoices = [

"white",

"red",

"pink",

"crimson",

"hotpink",

"green",

"aquamarine",

"blue"];

const squareGeometry = new THREE.PlaneGeometry(1, 1);

this.instancedGeometry = new THREE.InstancedBufferGeometry();

Object.keys(squareGeometry.attributes).forEach(attr => {

this.instancedGeometry.attributes[attr] = squareGeometry.attributes[attr];

});

this.instancedGeometry.index = squareGeometry.index;

this.instancedGeometry.maxInstancedCount = count;

for (let i = 0; i < count; i++) {

const phi = Math.random() \* Math.PI \* 2;

const i3 = 3 \* i;

randoms[i] = Math.random();

randoms1[i] = Math.random();

scales[i] = Math.random() \* 0.35;

const colorIndex = Math.floor(Math.random() \* colorChoices.length);

const color = new THREE.Color(colorChoices[colorIndex]);

colors[i3 + 0] = color.r;

colors[i3 + 1] = color.g;

colors[i3 + 2] = color.b;

speeds[i] = Math.random() \* this.parameters.max;

}

this.instancedGeometry.setAttribute(

"random",

new THREE.InstancedBufferAttribute(randoms, 1, false));

this.instancedGeometry.setAttribute(

"random1",

new THREE.InstancedBufferAttribute(randoms1, 1, false));

this.instancedGeometry.setAttribute(

"aScale",

new THREE.InstancedBufferAttribute(scales, 1, false));

this.instancedGeometry.setAttribute(

"aSpeed",

new THREE.InstancedBufferAttribute(speeds, 1, false));

this.instancedGeometry.setAttribute(

"aColor",

new THREE.InstancedBufferAttribute(colors, 3, false));

this.heart = new THREE.Mesh(this.instancedGeometry, this.heartMaterial);

console.log(this.heart);

this.scene.add(this.heart);

}

addToScene() {

this.addModel();

this.addHeart();

this.addSnow();

}

async addModel() {

this.model = await this.loadObj(

"https://assets.codepen.io/74321/heart.glb");

this.model.scale.set(0.01, 0.01, 0.01);

this.model.material = new THREE.MeshMatcapMaterial({

matcap: this.textureLoader.load(

"https://assets.codepen.io/74321/3.png",

() => {

gsap.to(this.model.scale, {

x: 0.35,

y: 0.35,

z: 0.35,

duration: 1.5,

ease: "Elastic.easeOut" });

}),

color: "#fff" });

this.scene.add(this.model);

}

addButton() {

this.audioBtn = document.querySelector("button");

this.audioBtn.addEventListener("click", () => {

this.audioBtn.disabled = true;

if (this.analyser) {

this.sound.play();

this.time.t0 = this.time.elapsed;

this.data = 0;

this.isRunning = true;

gsap.to(this.audioBtn, {

opacity: 0,

duration: 1,

ease: "power1.out" });

} else {

this.loadMusic().then(() => {

console.log("music loaded");

});

}

});

}

loadObj(path) {

const loader = new GLTFLoader();

return new Promise(resolve => {

loader.load(

path,

response => {

resolve(response.scene.children[0]);

},

xhr => {},

err => {

console.log(err);

});

});

}

loadMusic() {

return new Promise(resolve => {

const listener = new THREE.AudioListener();

this.camera.add(listener);

// create a global audio source

this.sound = new THREE.Audio(listener);

const audioLoader = new THREE.AudioLoader();

audioLoader.load(

"https://res.cloudinary.com/dmnxeusyw/video/upload/v1668310333/sharecs.net/music\_ji3iak.mp3",

buffer => {

this.sound.setBuffer(buffer);

this.sound.setLoop(false);

this.sound.setVolume(0.5);

this.sound.play();

this.analyser = new THREE.AudioAnalyser(this.sound, 32);

// get the average frequency of the sound

const data = this.analyser.getAverageFrequency();

this.isRunning = true;

this.t0 = this.time.elapsed;

resolve(data);

},

progress => {

gsap.to(this.audioBtn, {

opacity: () => 1 - progress.loaded / progress.total,

duration: 1,

ease: "power1.out" });

},

error => {

console.log(error);

});

});

}

addSnow() {

this.snowMaterial = new THREE.ShaderMaterial({

fragmentShader: document.getElementById("fragmentShader1").textContent,

vertexShader: document.getElementById("vertexShader1").textContent,

uniforms: {

uTime: { value: 0 },

uSize: { value: 0.3 },

uTex: {

value: new THREE.TextureLoader().load(

"https://assets.codepen.io/74321/heart.png") } },

depthWrite: false,

blending: THREE.AdditiveBlending,

transparent: true });

const count = 550;

const scales = new Float32Array(count \* 1);

const colors = new Float32Array(count \* 3);

const phis = new Float32Array(count);

const randoms = new Float32Array(count);

const randoms1 = new Float32Array(count);

const colorChoices = ["red", "pink", "hotpink", "green", "aquamarine", "blue"];

const squareGeometry = new THREE.PlaneGeometry(1, 1);

this.instancedGeometry = new THREE.InstancedBufferGeometry();

Object.keys(squareGeometry.attributes).forEach(attr => {

this.instancedGeometry.attributes[attr] = squareGeometry.attributes[attr];

});

this.instancedGeometry.index = squareGeometry.index;

this.instancedGeometry.maxInstancedCount = count;

for (let i = 0; i < count; i++) {

const phi = (Math.random() - 0.5) \* 10;

const i3 = 3 \* i;

phis[i] = phi;

randoms[i] = Math.random();

randoms1[i] = Math.random();

scales[i] = Math.random() \* 0.35;

const colorIndex = Math.floor(Math.random() \* colorChoices.length);

const color = new THREE.Color(colorChoices[colorIndex]);

colors[i3 + 0] = color.r;

colors[i3 + 1] = color.g;

colors[i3 + 2] = color.b;

}

this.instancedGeometry.setAttribute(

"phi",

new THREE.InstancedBufferAttribute(phis, 1, false));

this.instancedGeometry.setAttribute(

"random",

new THREE.InstancedBufferAttribute(randoms, 1, false));

this.instancedGeometry.setAttribute(

"random1",

new THREE.InstancedBufferAttribute(randoms1, 1, false));

this.instancedGeometry.setAttribute(

"aScale",

new THREE.InstancedBufferAttribute(scales, 1, false));

this.instancedGeometry.setAttribute(

"aColor",

new THREE.InstancedBufferAttribute(colors, 3, false));

this.snow = new THREE.Mesh(this.instancedGeometry, this.snowMaterial);

this.scene.add(this.snow);

}}

const world = new World({

canvas: document.querySelector("canvas.webgl"),

cameraPosition: { x: 0, y: 0, z: 4.5 } });

world.loop();

//# sourceURL=pen.js

</script>

</body>

</html>