tilt.js

<script src=”tilt.js”></script>

jQuery CDN Ekle

 <script

   src="https://code.jquery.com/jquery-3.5.1.js" integrity="sha256-QWo7LDvxbWT2tbbQ97B53yJnYU3WhH/C8ycbRAkjPDc=" crossorigin="anonymous">

  </script>

Hangi div’e ve section’a yazacaksan **data-tilt**

(function (factory) {

if (typeof define === 'function' && define.amd) {

// AMD. Register as an anonymous module.

define(['jquery'], factory);

} else if (typeof module === 'object' && module.exports) {

// Node/CommonJS

module.exports = function( root, jQuery ) {

if ( jQuery === undefined ) {

// require('jQuery') returns a factory that requires window to

// build a jQuery instance, we normalize how we use modules

// that require this pattern but the window provided is a noop

// if it's defined (how jquery works)

if ( typeof window !== 'undefined' ) {

jQuery = require('jquery');

}

else {

jQuery = require('jquery')(root);

}

}

factory(jQuery);

return jQuery;

};

} else {

// Browser globals

factory(jQuery);

}

}(function ($) {

$.fn.tilt = function (options) {

/\*\*

\* RequestAnimationFrame

\*/

const requestTick = function() {

if (this.ticking) return;

requestAnimationFrame(updateTransforms.bind(this));

this.ticking = true;

};

/\*\*

\* Bind mouse movement evens on instance

\*/

const bindEvents = function() {

const \_this = this;

$(this).on('mousemove', mouseMove);

$(this).on('mouseenter', mouseEnter);

if (this.settings.reset) $(this).on('mouseleave', mouseLeave);

if (this.settings.glare) $(window).on('resize', updateGlareSize.bind(\_this));

};

/\*\*

\* Set transition only on mouse leave and mouse enter so it doesn't influence mouse move transforms

\*/

const setTransition = function() {

if (this.timeout !== undefined) clearTimeout(this.timeout);

$(this).css({'transition': `${this.settings.speed}ms ${this.settings.easing}`});

if(this.settings.glare) this.glareElement.css({'transition': `opacity ${this.settings.speed}ms ${this.settings.easing}`});

this.timeout = setTimeout(() => {

$(this).css({'transition': ''});

if(this.settings.glare) this.glareElement.css({'transition': ''});

}, this.settings.speed);

};

/\*\*

\* When user mouse enters tilt element

\*/

const mouseEnter = function(event) {

this.ticking = false;

$(this).css({'will-change': 'transform'});

setTransition.call(this);

// Trigger change event

$(this).trigger("tilt.mouseEnter");

};

/\*\*

\* Return the x,y position of the mouse on the tilt element

\* @returns {{x: \*, y: \*}}

\*/

const getMousePositions = function(event) {

if (typeof(event) === "undefined") {

event = {

pageX: $(this).offset().left + $(this).outerWidth() / 2,

pageY: $(this).offset().top + $(this).outerHeight() / 2

};

}

return {x: event.pageX, y: event.pageY};

};

/\*\*

\* When user mouse moves over the tilt element

\*/

const mouseMove = function(event) {

this.mousePositions = getMousePositions(event);

requestTick.call(this);

};

/\*\*

\* When user mouse leaves tilt element

\*/

const mouseLeave = function() {

setTransition.call(this);

this.reset = true;

requestTick.call(this);

// Trigger change event

$(this).trigger("tilt.mouseLeave");

};

/\*\*

\* Get tilt values

\*

\* @returns {{x: tilt value, y: tilt value}}

\*/

const getValues = function() {

const width = $(this).outerWidth();

const height = $(this).outerHeight();

const left = $(this).offset().left;

const top = $(this).offset().top;

const percentageX = (this.mousePositions.x - left) / width;

const percentageY = (this.mousePositions.y - top) / height;

// x or y position inside instance / width of instance = percentage of position inside instance \* the max tilt value

const tiltX = ((this.settings.maxTilt / 2) - ((percentageX) \* this.settings.maxTilt)).toFixed(2);

const tiltY = (((percentageY) \* this.settings.maxTilt) - (this.settings.maxTilt / 2)).toFixed(2);

// angle

const angle = Math.atan2(this.mousePositions.x - (left+width/2),- (this.mousePositions.y - (top+height/2)) )\*(180/Math.PI);

// Return x & y tilt values

return {tiltX, tiltY, 'percentageX': percentageX \* 100, 'percentageY': percentageY \* 100, angle};

};

/\*\*

\* Update tilt transforms on mousemove

\*/

const updateTransforms = function() {

this.transforms = getValues.call(this);

if (this.reset) {

this.reset = false;

$(this).css('transform', `perspective(${this.settings.perspective}px) rotateX(0deg) rotateY(0deg)`);

// Rotate glare if enabled

if (this.settings.glare){

this.glareElement.css('transform', `rotate(180deg) translate(-50%, -50%)`);

this.glareElement.css('opacity', `0`);

}

return;

} else {

$(this).css('transform', `perspective(${this.settings.perspective}px) rotateX(${this.settings.disableAxis === 'x' ? 0 : this.transforms.tiltY}deg) rotateY(${this.settings.disableAxis === 'y' ? 0 : this.transforms.tiltX}deg) scale3d(${this.settings.scale},${this.settings.scale},${this.settings.scale})`);

// Rotate glare if enabled

if (this.settings.glare){

this.glareElement.css('transform', `rotate(${this.transforms.angle}deg) translate(-50%, -50%)`);

this.glareElement.css('opacity', `${this.transforms.percentageY \* this.settings.maxGlare / 100}`);

}

}

// Trigger change event

$(this).trigger("change", [this.transforms]);

this.ticking = false;

};

/\*\*

\* Prepare elements

\*/

const prepareGlare = function () {

const glarePrerender = this.settings.glarePrerender;

// If option pre-render is enabled we assume all html/css is present for an optimal glare effect.

if (!glarePrerender)

// Create glare element

$(this).append('<div class="js-tilt-glare"><div class="js-tilt-glare-inner"></div></div>');

// Store glare selector if glare is enabled

this.glareElementWrapper = $(this).find(".js-tilt-glare");

this.glareElement = $(this).find(".js-tilt-glare-inner");

// Remember? We assume all css is already set, so just return

if (glarePrerender) return;

// Abstracted re-usable glare styles

const stretch = {

'position': 'absolute',

'top': '0',

'left': '0',

'width': '100%',

'height': '100%',

};

// Style glare wrapper

this.glareElementWrapper.css(stretch).css({

'overflow': 'hidden',

'pointer-events': 'none',

});

// Style glare element

this.glareElement.css({

'position': 'absolute',

'top': '50%',

'left': '50%',

'background-image': `linear-gradient(0deg, rgba(255,255,255,0) 0%, rgba(255,255,255,1) 100%)`,

'width': `${$(this).outerWidth()\*2}`,

'height': `${$(this).outerWidth()\*2}`,

'transform': 'rotate(180deg) translate(-50%, -50%)',

'transform-origin': '0% 0%',

'opacity': '0',

});

};

/\*\*

\* Update glare on resize

\*/

const updateGlareSize = function () {

this.glareElement.css({

'width': `${$(this).outerWidth()\*2}`,

'height': `${$(this).outerWidth()\*2}`,

});

};

/\*\*

\* Public methods

\*/

$.fn.tilt.destroy = function() {

$(this).each(function () {

$(this).find('.js-tilt-glare').remove();

$(this).css({'will-change': '', 'transform': ''});

$(this).off('mousemove mouseenter mouseleave');

});

};

$.fn.tilt.getValues = function() {

const results = [];

$(this).each(function () {

this.mousePositions = getMousePositions.call(this);

results.push(getValues.call(this));

});

return results;

};

$.fn.tilt.reset = function() {

$(this).each(function () {

this.mousePositions = getMousePositions.call(this);

this.settings = $(this).data('settings');

mouseLeave.call(this);

setTimeout(() => {

this.reset = false;

}, this.settings.transition);

});

};

/\*\*

\* Loop every instance

\*/

return this.each(function () {

/\*\*

\* Default settings merged with user settings

\* Can be set trough data attributes or as parameter.

\* @type {\*}

\*/

this.settings = $.extend({

maxTilt: $(this).is('[data-tilt-max]') ? $(this).data('tilt-max') : 20,

perspective: $(this).is('[data-tilt-perspective]') ? $(this).data('tilt-perspective') : 300,

easing: $(this).is('[data-tilt-easing]') ? $(this).data('tilt-easing') : 'cubic-bezier(.03,.98,.52,.99)',

scale: $(this).is('[data-tilt-scale]') ? $(this).data('tilt-scale') : '1',

speed: $(this).is('[data-tilt-speed]') ? $(this).data('tilt-speed') : '400',

transition: $(this).is('[data-tilt-transition]') ? $(this).data('tilt-transition') : true,

disableAxis: $(this).is('[data-tilt-disable-axis]') ? $(this).data('tilt-disable-axis') : null,

axis: $(this).is('[data-tilt-axis]') ? $(this).data('tilt-axis') : null,

reset: $(this).is('[data-tilt-reset]') ? $(this).data('tilt-reset') : true,

glare: $(this).is('[data-tilt-glare]') ? $(this).data('tilt-glare') : false,

maxGlare: $(this).is('[data-tilt-maxglare]') ? $(this).data('tilt-maxglare') : 1,

}, options);

// Add deprecation warning & set disableAxis to deprecated axis setting

if(this.settings.axis !== null){

console.warn('Tilt.js: the axis setting has been renamed to disableAxis. See https://github.com/gijsroge/tilt.js/pull/26 for more information');

this.settings.disableAxis = this.settings.axis;

}

this.init = () => {

// Store settings

$(this).data('settings', this.settings);

// Prepare element

if(this.settings.glare) prepareGlare.call(this);

// Bind events

bindEvents.call(this);

};

// Init

this.init();

});

};

/\*\*

\* Auto load

\*/

$('[data-tilt]').tilt();

return true;

}));