Manual for the sPart system

Sindre Hauge Larsen, 2018/2019

Version 1.2, September 2019

Thank you for downloading the sPart system.

Contents

MIT License

What is it?

The sPart system is a 3D particle system for Gamemaker Studio 2 that provides you with a fast and easy way to create beautiful particle effects in 3D. It aims to be similar in use to the built-in 2D particle system in Gamemaker, and this documentation will assume the user has a basic familiarity with the built-in system.

How does it do it? Some technical info!

The particles are simulated entirely on the GPU via a vertex shader using the equations of motion. This is much, much faster than moving and drawing particles individually as sprites, and allows for drawing tens of thousands, or even hundreds of thousands of particles at a time. To be able to draw particles, we need to send a vertex buffer to the shader. The vertex buffer used by the sPart system is a bare-bones buffer that contains the following data per vertex:

1. The index of the particle, stored in three bytes. This allows for a maximum of 255x255x255 unique particles to be drawn per batch. This index is used to generate a unique seed for the noise function in the shader, giving each particle a unique trajectory.
2. The corner ID of the vertex, stored in the last byte. This can be 0, 1, 2, or 3. This is used to construct the local vertex position and texture coordinate.

The vertex buffer uses four bytes per vertex, and six vertices per particle. Multiple vertex buffers with differing numbers of particles can be created, so that the most fitting one can be used. This will be described later, in the function called spart_system_create. If the number of particles exceeds the maximum number of the vertex buffer, they will have to be split up into smaller batches.

We need to send some info about where to spawn particles to the shader. This is where the emitters come in handy. You can create an emitter and tell it how, where and how many particles it should emit per unit of time, and the sPart system sends this info to the vertex shader. Emitters are the most efficient when static, but they can be dynamically moved as well.

Particles can spawn other particles. This only works for one generation, particles spawned from other particles cannot spawn particles of their own. Particles can spawn other particles either each step or upon death. This uses a slightly different shader than the regular particles. This functionality is only available in the pro version!

Particles are by default billboarded images, but meshes can also be used to achieve fully 3D particles. The format for 3D mesh particles contains the following data per vertex:

1. Vertex position, stored in three bytes (one byte per dimension)
2. Particle index, stored in one byte (and so is capped to 255)
3. Texture coords stored in two bytes (one byte per dimension)
4. Vertex normals stored in two bytes (polar coordinate angles)

The mesh vertex buffer contains eight bytes per vertex. The number of vertices is uncapped, but for the sake of speed and memory usage, it is recommended to keep the vertex count low.

Particle system functions overview

Following is an overview of the various functions included in the sPart system, as well as a description of each.

spart_system_create(batchSizeArray)

Returns: Index of new particle system

Creates a new particle system, which is a container for the emitters created within the system. You must define the particle batch sizes in an array of ascending values, and the sPart system will at any time pick the batch size that fits best for your particles. For a dynamic particle system it's often useful to define multiple vertex batch sizes. Say there are only fifty particles on screen; if the minimum vertex batch is ten thousand, that is a lot of wasted vertices. On the opposite extreme, say there are ten thousand particles, but the maximum batch size is fifty – this would require a whole lot of draw calls to draw all particles, and would slow down your game!

Usage example:

- spart_system_create([50, 150, 500, 1500, 5000]); //<- this would let you draw a wide range of particles with as few draw calls as possible

spart_system_draw(partSystem, timeIncrement)

Returns: N/A

Draws a particle system. Must be called in draw event. You can set the world matrix with matrix_set(matrix_world, etc) to alter the position, orientation and scale of the entire particle system as a whole.

This script will also increment time and delete old emitters that are done emitting particles.

spart_system_destroy(partSystem)

Returns: N/A

Destroys the particle system and all emitters, particle types and vertex buffers it contains.

spart_system_set_shaders(partSystem, regShader, secondaryShader, meshShader)

Returns: N/A

Lets you define custom particle shaders for the given particle system

spart_system_set_dynamic_interval(partSystem, interval)

Returns: N/A

Set the interval for how often dynamic emitters may create new retired emitters. Higher interval means fewer new emitters and fewer draw calls, at the cost of movement artifacts.

Particle type functions overview

Following is a list of all particle type functions, as well as a description of each.

spart_type_create(partSystem)

Returns: Index of new particle type
Creates a new blank particle type. You can define the rules for this particle type with the following functions.

spart_type_destroy(partTypeInd)

Returns: N/A
Destroys a particle type. If this function isn't called, the particle type will live on for the entire duration of the game.

spart_type_life(partType, life_min, life_max)

Returns: N/A
Define the possible life span of each particle. A life span is randomly selected between the minimum and maximum.

spart_type_sprite(partType, sprite, image_speed, random)

Returns: N/A
Defines the sprite of a particle. The sprite can be animated, and you can set the animation speed. Set image_speed to -1 to stretch the animation to the particle's life span. This function will duplicate the sprite and transform it into a format the particle shader can use. Sprites whose dimensions are not powers of two will be resized to the nearest power of two. Animated sprites will be arranged one after another horizontally on the new sprite sheet. This new sprite is deleted when the particle type is destroyed. If the particle is a mesh particle, the sprite will be used as a texture. The texture may be animated as well.

spart_type_mesh(partType, model, numPerBatch)

Returns: N/A
Makes the particle draw a 3D mesh instead of a billboard. The meshes will be simulated in the same way as the billboarded particles, but will be drawn in full 3D and not in view-space. «Model» can be an index of a vertex buffer, a path to an .obj model or a path to a saved buffer. If a vertex buffer is supplied, it must be formatted in the following way:

1. 3D position	3x4 bytes
2. Normal	3x4 bytes
3. Texture coordinates	2x4 bytes
4. Colour	4x1 bytes

NumPerBatch cannot exceed 255

spart_type_mesh_rotation_axis(partType, x, y, z, axisDeviation)

Returns: N/A
A 3D mesh can rotate about an axis, and this axis is defined in this function. Give the axis to rotate around, as well as how far (in degrees) from this axis the rotation axis of the individual particles may deviate. Setting axisDeviation to 360 gives totally random rotation axes. The final rotation axis is also affected by the «radial» setting in spart_type_direction, which makes each particle face in its moving direction.

spart_type_mesh_lighting(partType, ambientCol, lightCol, lightDirX, lightDirY, lightDirZ)

Returns: N/A
Enables a directional light onto mesh particles. Only one directional light may be used with the built-in shaders, for more advanced lighting you will have to manually edit the shaders.

spart_type_direction(partType, xdir, ydir, zdir, dir_vary, radial)

Returns: N/A
Set the starting direction of the particle. Also lets you define the angle (in degrees) from the given direction the particle may randomly deviate. Setting relative to true modifies the starting direction to be relative to its spawning pos within the emitter. Particles that have been spawned by other particles will ignore their own direction, and instead keep moving in the direction of its creator, plus the direction variation.

spart_type_size (partType, size_min, size_max, size_incr, size_acc)

Returns: N/A
Sets the size of the particle. A starting size will be randomly selected between size_min and size_max, and the scale will be incremented by size_incr for each unit of time.

spart_type_orientation(partType, ang_min, ang_max, ang_incr, ang_acc, ang_relative)

Returns: N/A
Defines the angle and angular momentum (in degrees per unit of time) of the particle in screen-space. A starting angle will be randomly selected between ang_min and ang_max, and the angle will be incremented by ang_incr for each unit of time. Setting ang_relative to true will make the particle face in its screen-space moving direction, plus the given angle.

spart_type_speed(partType, speed_min, speed_max, speed_acc, speed_jerk)

Returns: N/A
Sets the speed of the particle. A starting speed will be randomly selected between speed_min and speed_max, and the speed will be incremented by speed_acc for each unit of time. The speed will also be incremented by speed_jerk for each unit of time squared, enabling more advanced movement patterns.

spart_type_gravity(partType, grav_amount, xdir, ydir, zdir)

Returns: N/A
Define the gravity vector and amount.

spart_type_colour_mix(partType, colour1, alpha1, colour2, alpha2)

Returns: N/A
Sets the colour of the particle to a random mix between the two given colours

spart_type_colour1/2/3/4(partType, colour1, alpha1, colour2, alpha2…etc)

Returns: N/A
Sets the colour of the particle to a linear blend of the given colours, stretched over the course of the particle’s life.

spart_type_blend(partType, additive)

Returns: N/A
Lets you toggle additive blend mode. Turning additive blend mode on will also disable zwriting. This is useful for, for example, fire effects.

spart_type_blend_ext(partType, src, dest, zwrite)

Returns: N/A
Lets you define source and destination blend modes, as well as give you control over the zwriting, for the particle.

spart_type_step(partType, num, childType)

Returns: N/A
Make the particle emit a given number of particles for each unit of time. Note that particles emitted this way cannot emit particles of their own.

spart_type_death(partType, num, childType)

Returns: N/A
Make the particle emit a given number of particles upon death. Note that particles emitted this way cannot emit particles of their own.

spart_type_save(partType, fname)

Returns: N/A
Saves your particle type to a file. This also saves any secondary particles.

spart_type_load(partSystem, fname)

Returns: Index of new particle type
Loads a particle type from file.

Particle emitter function overview

Following is a list of all particle emitter functions, as well as a description of each.

spart_emitter_create(partSystem)

Returns: Index of new emitter
Create a new inactive emitter. To activate the emitter, use spart_emitter_stream.

spart_emitter_stream(emitterInd, parttype, particleNumPerTime, lifeTime, dynamic)

Returns: N/A
Make the emitter emit the given number of particles per unit of time. You can define the emitters lifetime, or set it to -1 to use the default life span. Setting dynamic to true will let you change the emitter's orientation without altering the particles that have already been created. This effectively keeps a «log» of all previous emitter positions/orientations, so that existing particles are not retroactively altered.

spart_emitter_region(ind, M[16], xscale, yscale, zscale, shape, distribution, dynamic)

Returns: N/A
Change the orientation, scale, shape and distribution of the emitter.
Set dynamic to true if you'd like the particles that have already been created to finish drawing with the old settings.

M must be a matrix WITHOUT SCALING. This is important! Scaling must be supplied on its own in the next three arguments.
You can build an orientation matrix like this:
matrix_build(x, y, z, xrot, yrot, zrot, 1, 1, 1);

You can use the following constants for shape:
spart_shape_circle
spart_shape_cube
spart_shape_sphere
spart_shape_cylinder

You can use the following constants for distribution:
ps_distr_linear
ps_distr_gaussian //Not actually gaussian, but something that resembles it
ps_distr_invgaussian //Not actually inverse gaussian,, but something that resembles it

spart_emitter_mature(emitterInd)

Returns: N/A
This function will mature an emitter. This means that it will look like it has been emitting for quite a while, even though it just started. This function must be called after you're done defining the emitter, as all it really does is subtract the maximum life span of its particle types from the emitter's creation time.

spart_emitter_retire(ind)

Returns: N/A
Deletes the selected emitter from the active emitters list, and makes a new, special emitter that will only finish the old emitter's spawned particles without spawning new ones for its own. The new emitter will indicate the given emitter as its parent.

spart_emitter_destroy(ind)

Returns: N/A

Destroys the emitter, as well as any child emitters. This function will be unnecessary if you also delete the particle system with spart_system_destroy, as that function also deletes all emitters.

spart_particles_create(partSystem, x, y, z, regionRadius, parttype, number)

Returns: N/A

Creates a special type of emitter that only draws the given number of particles. Once it's done, it's deleted automatically in the spart_system_draw script.

spart_emitter_mesh_region(ind, M[16], xsize, ysize, zsize, emitterMesh, dynamic)

Returns: N/A

Change the orientation, scale and shape of the emitter. This particular script allows you to make an emitter that only emits particles within the confines of the given emitter mesh. emitterMesh must have been created beforehand using spart_emitter_mesh_preprocess on a model, like this:

```
emitterMesh = spart_emitter_mesh_preprocess(objPath, partNum, hollow);
```

Set dynamic to true if you'd like the particles that have already been created to finish drawing with the old settings.

M must be a matrix WITHOUT SCALING. This is important! Scaling must be supplied on its own in the next three arguments.

You can build an orientation matrix like this:

```
matrix_build(x, y, z, xrot, yrot, zrot, 1, 1, 1);
```

spart_emitter_mesh_preprocess(mesh, particleNum, hollow)

Returns: Array containing emitter mesh information

Preprocess the given mesh, creating all the possible particle positions within the mesh.

Mesh can either be a buffer using the standard format, or the path to an OBJ file.

Hollow mesh emitters are much faster to compute than non-hollow ones!

spart_emitter_mesh_save(emitterMesh, fname)

Returns: N/A

Save an emitter mesh to file

spart_emitter_mesh_load(fname)

Returns: Array containing emitter mesh information

Load an emitter mesh from file

spart_emitter_mesh_write_to_buffer(buffer, mesh)

Returns: N/A

Writes an emitter mesh to a buffer

spart_emitter_mesh_read_from_buffer(buffer)

Returns: Array containing emitter mesh information

Reads an emitter mesh from a buffer

Helper function overview

Here’s an overview of the helper scripts that are not meant for used by themselves, but are used in the other functions.

spart__init()

Returns: N/A
This script is global in scope, so it does not have to be called from anywhere. Initializes necessary macros, enums and data structures for the sPart system to work.

spart__get_uniform_index(shader)

Returns: Uniform index of the given shader
Uniforms are indexed in a grid so that the system only has to index them once.
If the shader has not been indexed previously, get the relevant shader uniforms.

spart__emitter_set_uniforms(uniformIndex, emitterIndex)

Returns: N/A
Sets the shader uniforms for the given emitter

spart__ type_set_uniforms(uniformIndex, partType)

Returns: N/A
Sets the shader uniforms for the given particle type.

spart__type_set_gpu_settings(partType)

Returns: N/A
Sets the GPU settings for the given particle type

spart__type_set_parent_uniforms(uniformIndex, partType, partNum)

Returns: N/A
Sets the shader uniforms for the given parent particle type. This is used for particles that are spawned each step or upon death of other particles.

spart__submit_vbuffs(partSystem, partType, partNum, uniformIndex)

Returns: N/A
Splits the particles up into batches and submits them to the GPU

spart__update_vbuffers(partSystem)

Returns: N/A
Makes sure the necessary vertex buffers exist.

spart__load_obj_to_buffer(fname)

Returns: Buffer index
Loads a .obj file into a buffer that is formatted in a way the sPart system can use. This is used for loading meshes in the spart_type_mesh function.

spart__read_obj_line(str)

Returns: Array containing the info in the string that's been separated by blank spaces

spart__read_obj_vertstring(str)

Returns: The vertex, normal and texture index of the vertex

spart__read_obj_face(faceList, str)

Returns: N/A
Adds the triangles in str to the facelist

spart__write_type_to_buffer(buffer, partType, depth)

Returns: N/A
Writes a particle type to the given buffer. Also writes any secondary particles.

spart__read_type_from_buffer(buffer, partSystem)

Returns: Index of new particle type
Loads a particle type from the given buffer.

spart__update_partnum(partEmitter)

Returns: N/A
Updates an emitter with the number of particles that will be rendered at any time.