Special Effects Documentation:

Billboard:
How does it work in theory?
In theory a billboard is always showing it's surface to the viewer. This is achieved by first inversing the ModelViewMAtrix and then taking a look at the lookVector. Inversing the ModelViewMAtrix transforms the camera position (4th column) to object space. We ignore the vector in the direction the billboard should be normal (usually the y-axis), call it "xzVetctor" and then get the angle between the lookVector and the xzVector.
For a spherical rotation we now rotate the billboard around its right vector with the given angle and we're done with the billboarding.

How is it implemented?
The implementation is quite similar to the theory. First of all we calculate the inverseModelViewMatrix and extract our lookVector(campos - objectpos in object space) from it.
We then define a xzLookVector that ignores the y-value (projection onto xz-plane).
With the angle-function of vec3 we can calculate the angle between our xzLookVec and the lookVec.
Finally we assign the value of the mat4.rotate-Function to our matrix. However in the code we have to also factor in that when viewed from the other side (thus lookVec[1] < 0) we need the negativeRightNormalVector of the billboard and calculate with this value, because we don't use the cross product for the rotation axis here which would handle the sign automatically. Using the cross product we have to be careful about the corner case where the product is of length zero.

Particle System?
How does it work in theory?
The particles are rendered using a technique called "Instancing". Instancing means, there is a single base mesh (a quad of two triangles in our case) of which multiple instances are rendered every frame. Instead of firing one OpenGL-drawcall per instance a dedicated instancing-drawcall is triggered once for all instances. Instance-attributes like the position which would be normally passed to the GPU via uniforms are streamed via buffers. This technique offers a significant performance boost when it comes to rendering a huge amount of instances of a plain mesh such as a particle.

How is it implemented?
To harness the instancing technique in WebGL first of all a common extension called "ANGLE" has to be retrieved. This extension is available on most of todays platforms. The positions of the centers of the particles, their vertices, colors and size are streamed to the particle shader via buffers. The "vertexAttribDivisor" function tells the gl-context how many values of a given buffer should be passed for each instance. "drawArraysInstanced" finally draws the particles. In the vertex shader the particles are billboarded, so that the always face the viewplane. This simple form of billboarding can be achieved by setting the 3x3-submatrix to the identity matrix. In the fragment shader the particle is renderd to a circle with

transparent edges. The particles have to be sorted from back to front (rendering order) on the CPU, so that blending works properly.