uniform vec3 iResolution; // viewport resolution (in pixels)  
uniform float iTime; // shader playback time (in seconds)  
uniform float iTimeDelta; // render time (in seconds)  
uniform int iFrame; // shader playback frame  
uniform float iChannelTime[4]; // channel playback time (in seconds)  
uniform vec3 iChannelResolution[4]; // channel resolution (in pixels)  
uniform vec4 iMouse; // mouse pixel coords. xy: current (if MLB down), zw: click  
uniform samplerXX iChannel0..3; // input channel. XX = 2D/Cube  
uniform vec4 iDate; // (year, month, day, time in seconds)  
uniform float iSampleRate; // sound sample rate (i.e., 44100)

// Created by inigo quilez - iq/2019

// License Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.

//

//

// An animation test - a happy and blobby creature jumping and

// looking around. It gets off-model very often, but it looks

// good enough I think.

//

// Making-of and related math/shader/art explanations (6 hours

// long): https://www.youtube.com/watch?v=Cfe5UQ-1L9Q

//

// Video capture: https://www.youtube.com/watch?v=s\_UOFo2IULQ

//

// You can buy a metal print of this shader here:

// https://www.redbubble.com/i/metal-print/Happy-Jumping-by-InigoQuilez/43594745.0JXQP

#if HW\_PERFORMANCE==0

#define AA 1

#else

#define AA 2 // Set AA to 1 if your machine is too slow

#endif

//------------------------------------------------------------------

// http://iquilezles.org/www/articles/smin/smin.htm

float smin( float a, float b, float k )

{

float h = max(k-abs(a-b),0.0);

return min(a, b) - h\*h\*0.25/k;

}

// http://iquilezles.org/www/articles/smin/smin.htm

vec2 smin( vec2 a, vec2 b, float k )

{

float h = clamp( 0.5+0.5\*(b.x-a.x)/k, 0.0, 1.0 );

return mix( b, a, h ) - k\*h\*(1.0-h);

}

// http://iquilezles.org/www/articles/smin/smin.htm

float smax( float a, float b, float k )

{

float h = max(k-abs(a-b),0.0);

return max(a, b) + h\*h\*0.25/k;

}

// http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

float sdSphere( vec3 p, float s )

{

return length(p)-s;

}

// http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm

float sdEllipsoid( in vec3 p, in vec3 r ) // approximated

{

float k0 = length(p/r);

float k1 = length(p/(r\*r));

return k0\*(k0-1.0)/k1;

}

vec2 sdStick(vec3 p, vec3 a, vec3 b, float r1, float r2) // approximated

{

vec3 pa = p-a, ba = b-a;

float h = clamp( dot(pa,ba)/dot(ba,ba), 0.0, 1.0 );

return vec2( length( pa - ba\*h ) - mix(r1,r2,h\*h\*(3.0-2.0\*h)), h );

}

// http://iquilezles.org/www/articles/smin/smin.htm

vec4 opU( vec4 d1, vec4 d2 )

{

return (d1.x<d2.x) ? d1 : d2;

}

//------------------------------------------------------------------

#define ZERO (min(iFrame,0))

//------------------------------------------------------------------

float href;

float hsha;

vec4 map( in vec3 pos, float atime )

{

hsha = 1.0;

float t1 = fract(atime);

float t4 = abs(fract(atime\*0.5)-0.5)/0.5;

float p = 4.0\*t1\*(1.0-t1);

float pp = 4.0\*(1.0-2.0\*t1); // derivative of p

vec3 cen = vec3( 0.5\*(-1.0 + 2.0\*t4),

pow(p,2.0-p) + 0.1,

floor(atime) + pow(t1,0.7) -1.0 );

// body

vec2 uu = normalize(vec2( 1.0, -pp ));

vec2 vv = vec2(-uu.y, uu.x);

float sy = 0.5 + 0.5\*p;

float compress = 1.0-smoothstep(0.0,0.4,p);

sy = sy\*(1.0-compress) + compress;

float sz = 1.0/sy;

vec3 q = pos - cen;

float rot = -0.25\*(-1.0 + 2.0\*t4);

float rc = cos(rot);

float rs = sin(rot);

q.xy = mat2x2(rc,rs,-rs,rc)\*q.xy;

vec3 r = q;

href = q.y;

q.yz = vec2( dot(uu,q.yz), dot(vv,q.yz) );

vec4 res = vec4( sdEllipsoid( q, vec3(0.25, 0.25\*sy, 0.25\*sz) ), 2.0, 0.0, 1.0 );

if( res.x-1.0 < pos.y ) // bounding volume

{

float t2 = fract(atime+0.8);

float p2 = 0.5-0.5\*cos(6.2831\*t2);

r.z += 0.05-0.2\*p2;

r.y += 0.2\*sy-0.2;

vec3 sq = vec3( abs(r.x), r.yz );

// head

vec3 h = r;

float hr = sin(0.791\*atime);

hr = 0.7\*sign(hr)\*smoothstep(0.5,0.7,abs(hr));

h.xz = mat2x2(cos(hr),sin(hr),-sin(hr),cos(hr))\*h.xz;

vec3 hq = vec3( abs(h.x), h.yz );

float d = sdEllipsoid( h-vec3(0.0,0.20,0.02), vec3(0.08,0.2,0.15) );

float d2 = sdEllipsoid( h-vec3(0.0,0.21,-0.1), vec3(0.20,0.2,0.20) );

d = smin( d, d2, 0.1 );

res.x = smin( res.x, d, 0.1 );

// belly wrinkles

{

float yy = r.y-0.02-2.5\*r.x\*r.x;

res.x += 0.001\*sin(yy\*120.0)\*(1.0-smoothstep(0.0,0.1,abs(yy)));

}

// arms

{

vec2 arms = sdStick( sq, vec3(0.18-0.06\*hr\*sign(r.x),0.2,-0.05), vec3(0.3+0.1\*p2,-0.2+0.3\*p2,-0.15), 0.03, 0.06 );

res.xz = smin( res.xz, arms, 0.01+0.04\*(1.0-arms.y)\*(1.0-arms.y)\*(1.0-arms.y) );

}

// ears

{

float t3 = fract(atime+0.9);

float p3 = 4.0\*t3\*(1.0-t3);

vec2 ear = sdStick( hq, vec3(0.15,0.32,-0.05), vec3(0.2+0.05\*p3,0.2+0.2\*p3,-0.07), 0.01, 0.04 );

res.xz = smin( res.xz, ear, 0.01 );

}

// mouth

{

d = sdEllipsoid( h-vec3(0.0,0.15+4.0\*hq.x\*hq.x,0.15), vec3(0.1,0.04,0.2) );

res.w = 0.3+0.7\*clamp( d\*150.0,0.0,1.0);

res.x = smax( res.x, -d, 0.03 );

}

// legs

{

float t6 = cos(6.2831\*(atime\*0.5+0.25));

float ccc = cos(1.57\*t6\*sign(r.x));

float sss = sin(1.57\*t6\*sign(r.x));

vec3 base = vec3(0.12,-0.07,-0.1); base.y -= 0.1/sy;

vec2 legs = sdStick( sq, base, base + vec3(0.2,-ccc,sss)\*0.2, 0.04, 0.07 );

res.xz = smin( res.xz, legs, 0.07 );

}

// eye

{

float blink = pow(0.5+0.5\*sin(2.1\*iTime),20.0);

float eyeball = sdSphere(hq-vec3(0.08,0.27,0.06),0.065+0.02\*blink);

res.x = smin( res.x, eyeball, 0.03 );

vec3 cq = hq-vec3(0.1,0.34,0.08);

cq.xy = mat2x2(0.8,0.6,-0.6,0.8)\*cq.xy;

d = sdEllipsoid( cq, vec3(0.06,0.03,0.03) );

res.x = smin( res.x, d, 0.03 );

float eo = 1.0-0.5\*smoothstep(0.01,0.04,length((hq.xy-vec2(0.095,0.285))\*vec2(1.0,1.1)));

res = opU( res, vec4(sdSphere(hq-vec3(0.08,0.28,0.08),0.060),3.0,0.0,eo));

res = opU( res, vec4(sdSphere(hq-vec3(0.075,0.28,0.102),0.0395),4.0,0.0,1.0));

}

}

// ground

float fh = -0.1 - 0.05\*(sin(pos.x\*2.0)+sin(pos.z\*2.0));

float t5f = fract(atime+0.05);

float t5i = floor(atime+0.05);

float bt4 = abs(fract(t5i\*0.5)-0.5)/0.5;

vec2 bcen = vec2( 0.5\*(-1.0+2.0\*bt4),t5i+pow(t5f,0.7)-1.0 );

float k = length(pos.xz-bcen);

float tt = t5f\*15.0-6.2831 - k\*3.0;

fh -= 0.1\*exp(-k\*k)\*sin(tt)\*exp(-max(tt,0.0)/2.0)\*smoothstep(0.0,0.01,t5f);

float d = pos.y - fh;

// bubbles

{

vec3 vp = vec3( mod(abs(pos.x),3.0)-1.5,pos.y,mod(pos.z+1.5,3.0)-1.5);

vec2 id = vec2( floor(pos.x/3.0), floor((pos.z+1.5)/3.0) );

float fid = id.x\*11.1 + id.y\*31.7;

float fy = fract(fid\*1.312+atime\*0.1);

float y = -1.0+4.0\*fy;

vec3 rad = vec3(0.7,1.0+0.5\*sin(fid),0.7);

rad -= 0.1\*(sin(pos.x\*3.0)+sin(pos.y\*4.0)+sin(pos.z\*5.0));

float siz = 4.0\*fy\*(1.0-fy);

float d2 = sdEllipsoid( vp-vec3(0.5,y,0.0), siz\*rad );

d2 -= 0.03\*smoothstep(-1.0,1.0,sin(18.0\*pos.x)+sin(18.0\*pos.y)+sin(18.0\*pos.z));

d2 \*= 0.6;

d2 = min(d2,2.0);

d = smin( d, d2, 0.32 );

if( d<res.x ) { res = vec4(d,1.0,0.0,1.0); hsha=sqrt(siz); }

}

// candy

{

float fs = 5.0;

vec3 qos = fs\*vec3(pos.x, pos.y-fh, pos.z );

vec2 id = vec2( floor(qos.x+0.5), floor(qos.z+0.5) );

vec3 vp = vec3( fract(qos.x+0.5)-0.5,qos.y,fract(qos.z+0.5)-0.5);

vp.xz += 0.1\*cos( id.x\*130.143 + id.y\*120.372 + vec2(0.0,2.0) );

float den = sin(id.x\*0.1+sin(id.y\*0.091))+sin(id.y\*0.1);

float fid = id.x\*0.143 + id.y\*0.372;

float ra = smoothstep(0.0,0.1,den\*0.1+fract(fid)-0.95);

d = sdSphere( vp, 0.35\*ra )/fs;

if( d<res.x ) res = vec4(d,5.0,qos.y,1.0);

}

return res;

}

vec4 castRay( in vec3 ro, in vec3 rd, float time )

{

vec4 res = vec4(-1.0,-1.0,0.0,1.0);

float tmin = 0.5;

float tmax = 20.0;

#if 1

// raytrace bounding plane

float tp = (3.5-ro.y)/rd.y;

if( tp>0.0 ) tmax = min( tmax, tp );

#endif

// raymarch scene

float t = tmin;

for( int i=0; i<256 && t<tmax; i++ )

{

vec4 h = map( ro+rd\*t, time );

if( abs(h.x)<(0.0005\*t) )

{

res = vec4(t,h.yzw);

break;

}

t += h.x;

}

return res;

}

// http://iquilezles.org/www/articles/rmshadows/rmshadows.htm

float calcSoftshadow( in vec3 ro, in vec3 rd, float time )

{

float res = 1.0;

float tmax = 12.0;

#if 1

float tp = (3.5-ro.y)/rd.y; // raytrace bounding plane

if( tp>0.0 ) tmax = min( tmax, tp );

#endif

float t = 0.02;

for( int i=0; i<50; i++ )

{

float h = map( ro + rd\*t, time ).x;

res = min( res, mix(1.0,16.0\*h/t, hsha) );

t += clamp( h, 0.05, 0.40 );

if( res<0.005 || t>tmax ) break;

}

return clamp( res, 0.0, 1.0 );

}

// http://iquilezles.org/www/articles/normalsSDF/normalsSDF.htm

vec3 calcNormal( in vec3 pos, float time )

{

#if 0

vec2 e = vec2(1.0,-1.0)\*0.5773\*0.001;

return normalize( e.xyy\*map( pos + e.xyy, time ).x +

e.yyx\*map( pos + e.yyx, time ).x +

e.yxy\*map( pos + e.yxy, time ).x +

e.xxx\*map( pos + e.xxx, time ).x );

#else

// inspired by tdhooper and klems - a way to prevent the compiler from inlining map() 4 times

vec3 n = vec3(0.0);

for( int i=ZERO; i<4; i++ )

{

vec3 e = 0.5773\*(2.0\*vec3((((i+3)>>1)&1),((i>>1)&1),(i&1))-1.0);

n += e\*map(pos+0.001\*e,time).x;

}

return normalize(n);

#endif

}

float calcOcclusion( in vec3 pos, in vec3 nor, float time )

{

float occ = 0.0;

float sca = 1.0;

for( int i=ZERO; i<5; i++ )

{

float h = 0.01 + 0.11\*float(i)/4.0;

vec3 opos = pos + h\*nor;

float d = map( opos, time ).x;

occ += (h-d)\*sca;

sca \*= 0.95;

}

return clamp( 1.0 - 2.0\*occ, 0.0, 1.0 );

}

vec3 render( in vec3 ro, in vec3 rd, float time )

{

// sky dome

vec3 col = vec3(0.5, 0.8, 0.9) - max(rd.y,0.0)\*0.5;

// sky clouds

vec2 uv = 1.5\*rd.xz/rd.y;

float cl = 1.0\*(sin(uv.x)+sin(uv.y)); uv \*= mat2(0.8,0.6,-0.6,0.8)\*2.1;

cl += 0.5\*(sin(uv.x)+sin(uv.y));

col += 0.1\*(-1.0+2.0\*smoothstep(-0.1,0.1,cl-0.4));

// sky horizon

col = mix( col, vec3(0.5, 0.7, .9), exp(-10.0\*max(rd.y,0.0)) );

// scene geometry

vec4 res = castRay(ro,rd, time);

if( res.y>-0.5 )

{

float t = res.x;

vec3 pos = ro + t\*rd;

vec3 nor = calcNormal( pos, time );

vec3 ref = reflect( rd, nor );

float focc = res.w;

// material

col = vec3(0.2);

float ks = 1.0;

if( res.y>4.5 ) // candy

{

col = vec3(0.14,0.048,0.0);

vec2 id = floor(5.0\*pos.xz+0.5);

col += 0.036\*cos((id.x\*11.1+id.y\*37.341) + vec3(0.0,1.0,2.0) );

col = max(col,0.0);

focc = clamp(4.0\*res.z,0.0,1.0);

}

else if( res.y>3.5 ) // eyeball

{

col = vec3(0.0);

}

else if( res.y>2.5 ) // iris

{

col = vec3(0.4);

}

else if( res.y>1.5 ) // body

{

col = mix(vec3(0.144,0.09,0.0036),vec3(0.36,0.1,0.04),res.z\*res.z);

col = mix(col,vec3(0.14,0.09,0.06)\*2.0, (1.0-res.z)\*smoothstep(-0.15, 0.15, -href));

}

else // terrain

{

// base green

col = vec3(0.05,0.09,0.02);

float f = 0.2\*(-1.0+2.0\*smoothstep(-0.2,0.2,sin(18.0\*pos.x)+sin(18.0\*pos.y)+sin(18.0\*pos.z)));

col += f\*vec3(0.06,0.06,0.02);

ks = 0.5 + pos.y\*0.15;

// footprints

vec2 mp = vec2(pos.x-0.5\*(mod(floor(pos.z+0.5),2.0)\*2.0-1.0), fract(pos.z+0.5)-0.5 );

float mark = 1.0-smoothstep(0.1, 0.5, length(mp));

mark \*= smoothstep(0.0, 0.1, floor(time) - floor(pos.z+0.5) );

col \*= mix( vec3(1.0), vec3(0.5,0.5,0.4), mark );

ks \*= 1.0-0.5\*mark;

}

// lighting (sun, sky, bounce, back, sss)

float occ = calcOcclusion( pos, nor, time )\*focc;

float fre = clamp(1.0+dot(nor,rd),0.0,1.0);

vec3 sun\_lig = normalize( vec3(0.6, 0.35, 0.5) );

float sun\_dif = clamp(dot( nor, sun\_lig ), 0.0, 1.0 );

vec3 sun\_hal = normalize( sun\_lig-rd );

float sun\_sha = calcSoftshadow( pos, sun\_lig, time );

float sun\_spe = ks\*pow(clamp(dot(nor,sun\_hal),0.0,1.0),8.0)\*sun\_dif\*(0.04+0.96\*pow(clamp(1.0+dot(sun\_hal,rd),0.0,1.0),5.0));

float sky\_dif = sqrt(clamp( 0.5+0.5\*nor.y, 0.0, 1.0 ));

float sky\_spe = ks\*smoothstep( 0.0, 0.5, ref.y )\*(0.04+0.96\*pow(fre,4.0));

float bou\_dif = sqrt(clamp( 0.1-0.9\*nor.y, 0.0, 1.0 ))\*clamp(1.0-0.1\*pos.y,0.0,1.0);

float bac\_dif = clamp(0.1+0.9\*dot( nor, normalize(vec3(-sun\_lig.x,0.0,-sun\_lig.z))), 0.0, 1.0 );

float sss\_dif = fre\*sky\_dif\*(0.25+0.75\*sun\_dif\*sun\_sha);

vec3 lin = vec3(0.0);

lin += sun\_dif\*vec3(8.10,6.00,4.20)\*vec3(sun\_sha,sun\_sha\*sun\_sha\*0.5+0.5\*sun\_sha,sun\_sha\*sun\_sha);

lin += sky\_dif\*vec3(0.50,0.70,1.00)\*occ;

lin += bou\_dif\*vec3(0.20,0.70,0.10)\*occ;

lin += bac\_dif\*vec3(0.45,0.35,0.25)\*occ;

lin += sss\_dif\*vec3(3.25,2.75,2.50)\*occ;

col = col\*lin;

col += sun\_spe\*vec3(9.90,8.10,6.30)\*sun\_sha;

col += sky\_spe\*vec3(0.20,0.30,0.65)\*occ\*occ;

col = pow(col,vec3(0.8,0.9,1.0) );

// fog

col = mix( col, vec3(0.5,0.7,0.9), 1.0-exp( -0.0001\*t\*t\*t ) );

}

return col;

}

mat3 setCamera( in vec3 ro, in vec3 ta, float cr )

{

vec3 cw = normalize(ta-ro);

vec3 cp = vec3(sin(cr), cos(cr),0.0);

vec3 cu = normalize( cross(cw,cp) );

vec3 cv = ( cross(cu,cw) );

return mat3( cu, cv, cw );

}

void mainImage( out vec4 fragColor, in vec2 fragCoord )

{

vec3 tot = vec3(0.0);

#if AA>1

for( int m=ZERO; m<AA; m++ )

for( int n=ZERO; n<AA; n++ )

{

// pixel coordinates

vec2 o = vec2(float(m),float(n)) / float(AA) - 0.5;

vec2 p = (-iResolution.xy + 2.0\*(fragCoord+o))/iResolution.y;

// time coordinate (motion blurred, shutter=0.5)

float d = 0.5+0.5\*sin(fragCoord.x\*147.0)\*sin(fragCoord.y\*131.0);

float time = iTime - 0.5\*(1.0/24.0)\*(float(m\*AA+n)+d)/float(AA\*AA);

#else

vec2 p = (-iResolution.xy + 2.0\*fragCoord)/iResolution.y;

float time = iTime;

#endif

time += -2.6;

time \*= 0.9;

// camera

float cl = sin(0.5\*time);

float an = 1.57 + 0.7\*sin(0.15\*time);

vec3 ta = vec3( 0.0, 0.65, -0.6+time\*1.0 - 0.4\*cl);

vec3 ro = ta + vec3( 1.3\*cos(an), -0.250, 1.3\*sin(an) );

float ti = fract(time-0.15);

ti = 4.0\*ti\*(1.0-ti);

ta.y += 0.15\*ti\*ti\*(3.0-2.0\*ti)\*smoothstep(0.4,0.9,cl);

// camera bounce

float t4 = abs(fract(time\*0.5)-0.5)/0.5;

float bou = -1.0 + 2.0\*t4;

ro += 0.06\*sin(time\*12.0+vec3(0.0,2.0,4.0))\*smoothstep( 0.85, 1.0, abs(bou) );

// camera-to-world rotation

mat3 ca = setCamera( ro, ta, 0.0 );

// ray direction

vec3 rd = ca \* normalize( vec3(p,1.8) );

// render

vec3 col = render( ro, rd, time );

// color grading

col = col\*vec3(1.11,0.89,0.79);

// compress

col = 1.35\*col/(1.0+col);

// gamma

col = pow( col, vec3(0.4545) );

tot += col;

#if AA>1

}

tot /= float(AA\*AA);

#endif

// s-surve

tot = clamp(tot,0.0,1.0);

tot = tot\*tot\*(3.0-2.0\*tot);

// vignetting

vec2 q = fragCoord/iResolution.xy;

tot \*= 0.5 + 0.5\*pow(16.0\*q.x\*q.y\*(1.0-q.x)\*(1.0-q.y),0.25);

// output

fragColor = vec4( tot, 1.0 );

}