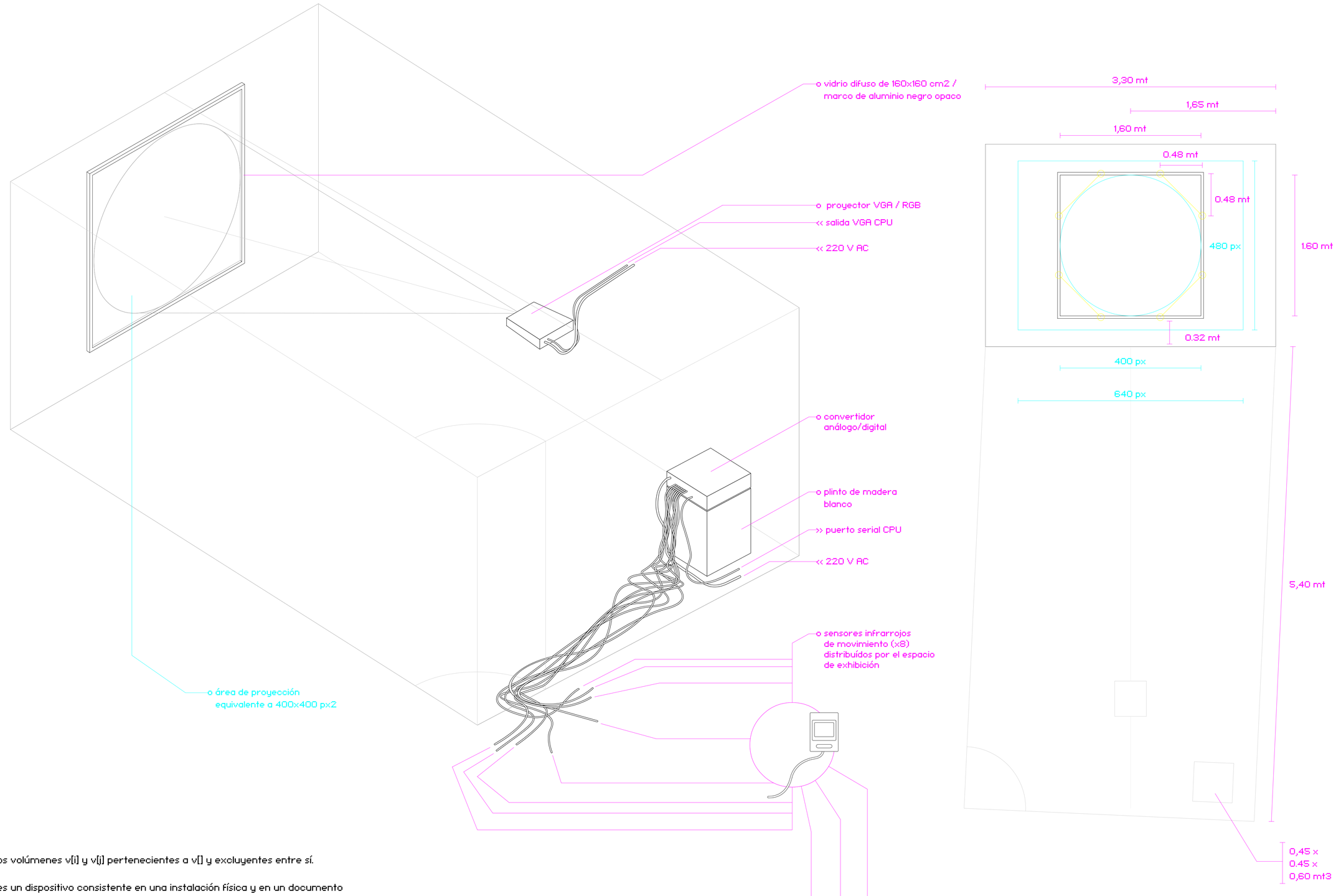


Dados dos volúmenes v[i] y v[j] pertenecientes a v[] y excluyentes entre sí.

crc/cw es un dispositivo consistente en una instalación física y en un documento publicado en internet, presentado como prototipo de posible objeto de arte, desde el contexto local hacia uno de expansión global o, más específicamente, vecinal.

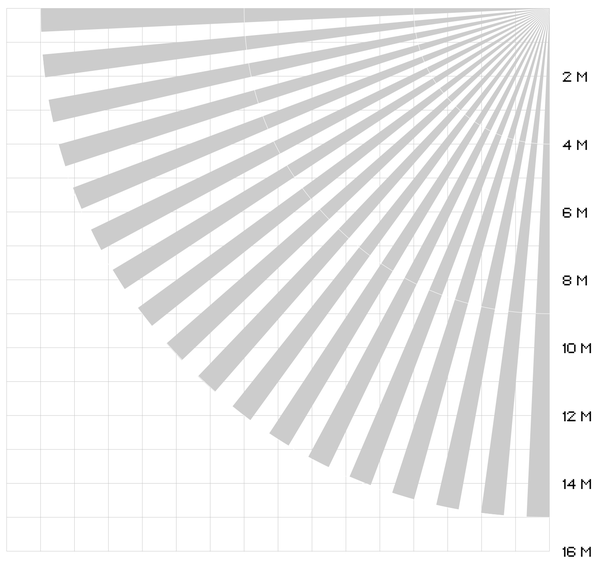
crc/cw es una propuesta de obra basada y condicionada por una investigación sobre los procedimientos que le darían especificidad a la relación arte y tecnologías digitales, inscrita al amparo de la pregunta por la pertinencia de la tecnologización progresiva en el contexto de la producción local, periférica a los centros globales de desarrollo tecnológico.

crc/cw es un objeto tecnológico limítrofe compuesto de 4 módulos:
-un módulo de entrada: 8 sensores infrarrojos de movimiento distribuídos en V[i] con los que se obtiene una representación de un byte correspondiente a la circulación en el espacio de exhibición.
-un módulo de proceso: dispositivo de digitalización y uno de representación de datos, consistente en un sistema de ciclos/patrones temporales, gráficos y sonoros autónomos que sufren alteraciones en su despliegue dependiendo de la excitabilidad/irritabilidad del sistema.
-un módulo de salida/despliegue audiovisual: emplazado en v[j] es un dispositivo no instanciable de espacialización/representación que opera como proyección espacial de v[i] en v[j].
-un módulo de registro: como crc/cw sólo realiza registros en RAM sólo se dispondrá del acceso a su propio diagrama como registro.

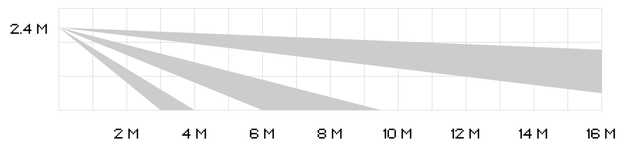
crc/cw opera como un sistema de control de feedback (como circuito o servomecanismo), vale decir, como un sistema cerrado de retroacción que mantiene una relativa estabilidad ante perturbaciones externas o internas.

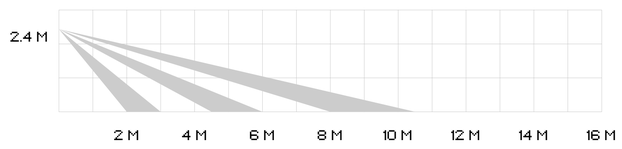
crc/cw está concebido, entonces, como un modelo implícitamente virtual -en tanto existe en su diagrama-, y desarrollado a partir de la articulación de elementos y operaciones precarias respecto del paradigma definido por el uso de nuevas tecnologías, siendo presentado como un proceso de especulación sobre el ruido/descalce que produce la retroalimentación de sistemas físicos y digitales, como un campo posible de construcción de sentido.

Cobertura Horizontal



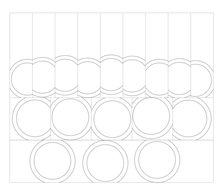
Cobertura Vertical (max / mín)





Volumetría

90º
34 zonas de corte
3 planos



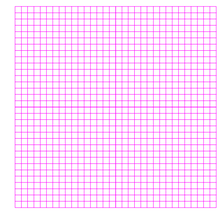
coyarzun:crc/cw:
diag_001:
www.error404.cl/crc_cw

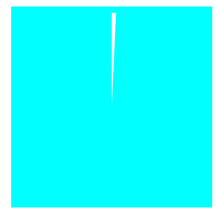
```
global oktopus

on prepareMovie me
  cursor 200
  puppetTempo 999
  oktopus = new(script "oktopus_device")
  add the actorList, oktopus
end

on stopMovie
  oktopus.remove()
  the actorList = []
end

on keyUp me
  oktopus.updatek()
end

on dec2bin n
  bin = ""
  repeat with i = 7 down to 0
    pow = integer(power(2, i))
    div = integer(n/pow)
    if div >= 1 then
      bin = bin&"1"
      n = n - pow
    else
      bin = bin&"0"
    end if
  end repeat
  return bin
end

on dec2rad n
  return(n*PI/180)
end
----------
--oktopus_device
property serial_obj, keyboard_obj, feedback_control_system
property input_type
property bin_string

on new me
  serial_obj = new(script "serial_obj", "COM1", 19200, "n", 8, 1)
  keyboard_obj = new(script "keyboard_obj")
  feedback_control_system = new(script "feedback_control_system", 400)
  input_type = "serial"
  return me
end

on config me, option
  input_type = option
end

on remove me
  serial_obj.remove()
end

on stepFrame me
  case input_type of:
    "serial":
      serial_obj.updatez()
      bin_string = serial_obj.bin_string
    "keyboard":
      bin_string = keyboard_obj.bin_string
  end case
  feedback_control_system.updatez(bin_string)
end

on updatek me
  case the keyCode of
    18:
      me.config("serial")
    19:
      me.config("keyboard")
    otherwise:
      if input_type = "keyboard" then
        keyboard_obj.updatez()
      end if
  end case
end
--------
--serial_obj
property serial_port
property input_dec, bin_string

on new me, port_name, baud_rate, parity, data_bits, stop_bits
  openxlib "SerialXtra.x32"
  input_dec = 0
  bin_string = "00000000"
  serial_port = new(xtra "SerialXtra")
  serial_port.openPort(port_name)
  serial_port.SetProtocol(baud_rate, parity, data_bits, stop_bits)
  return me
end

on remove me
  serial_port.ClosePort()
  serial_port = 0
  closexlib
end

on updatez me
  if serial_port.IsPortOpen() then
    buffer = serial_port.ReadBuffer()
    if(buffer.count()>0) then
      input_dec = buffer[1]
      bin_string = dec2bin(input_dec)
    end if
  else
    input_dec = 0
    bin_string = "00000000"
  end if
end
--------
--keyboard_obj
property input_dec, bin_string

on new me
  input_dec = 0
  bin_string = "00000000"
  return me
end

on updatez me
  input_dec = 0
  if the keyCode = 49 then
    input_dec = random(255)
  else if the keyCode = 8 then
    input_dec = 0
  end if
  bin_string = dec2bin(input_dec)
end

--------
--circle
property cx, cy, r

on new me, x, y, rr
  cx = x
  cy = y
  r = rr
  return me
end
```

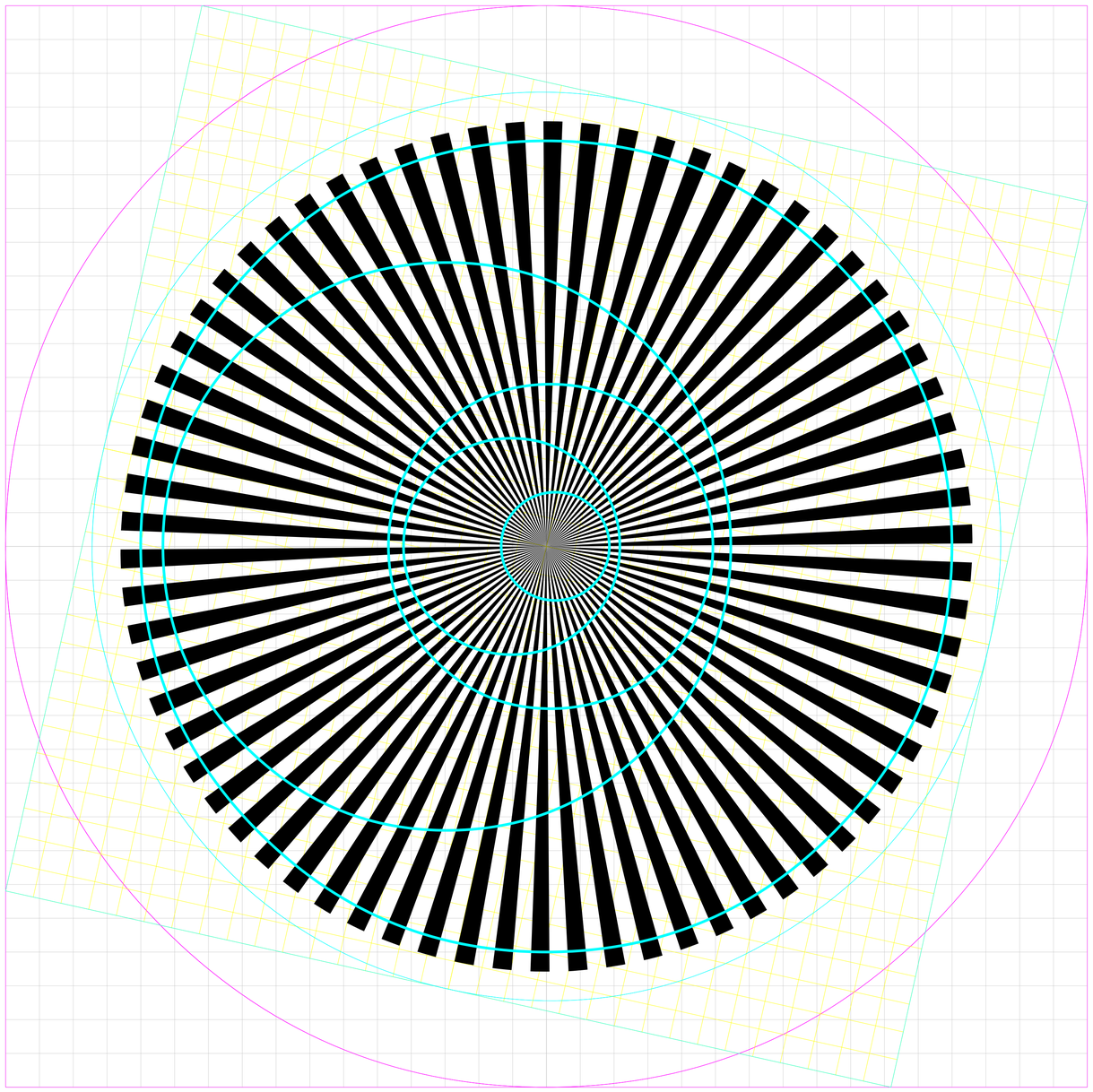
```
--feedback_control_system
property ocular, feedback, zoniko

on new me, s
  ocular = new(script "ocular_device", s)
  feedback = new(script "img_feedback", s)
  zoniko = new(script "zoniko", s)
  return me
end

on updatez me, bin_string
  feedback.updatez(bin_string)
  ocular.updatez(bin_string)
  zoniko.updatez(feedback)
end
--------
--ocular_device
property offscreenBuffer, grid_img, mzk_img
property side, w, h
property anglez, x_offzet, y_offzet
property srcImg, destRect, srcRect, blendlev, inklev, kolor
property cuad, roto
property activity_slide, friction, old_bin
property zones, swap_cond

on new me, s
  side = s
  w = s
  h = s
  offscreenBuffer = image(w, h, 24)
  grid_img = member("grid").image
  mzk_img = new(script "mzk", side)
  roto = new(script "roto", side, 1)
  zones = new(script "beam_zone", side)
  kolor = new(script "kolor_cycling", 1)
  destRect = new(script "kuad")
  blendlev = 255
  inklev = 36
  friction = .01
  activity_slide = 1
  swap_cond = false
  return me
end

on updatez me, bin_string
  to_activity = (bin_string <> "00000000")*1
  activity_slide = activity_slide + (to_activity-activity_slide)*friction
  if(old_bin<>bin_string) then
    kolor.kolor = rgb(0,0,0)
  else
    kolor.cycle(1)
  end if
  --
  anglez = anglez + PI/30
  l = side/(abs(sin(anglez))+abs(cos(anglez)))
  r = sqrt(float((l-1)*(l-1)/2))
  --
  offscreenBuffer.fill(offscreenBuffer.rect, rgb(255, 255, 255))
  --
  destRect.updatez(r*activity_slide, anglez, side-1)
  srcImg = grid_img
  srcRect = srcImg.rect
  offscreenBuffer.copyPixels(srcImg, destRect.rekt, srcRect, \
                             [#blendLevel:blendlev, #ink:0, #color:kolor.kolor])
  --
  if(swap_cond) then
    aux = 9
  else
    aux = 5
  end if
  repeat with i=1 to 4
    if(char (aux-i) of bin_string = "1")then
      srcImg = zones[i]
      srcRect = srcImg.rect
      offscreenBuffer.copyPixels(srcImg, destRect.rekt, srcRect, \
                                 [#blendLevel:blendlev, #ink:5, #color:kolor.kolor])
    end if
  end repeat
  swap_cond = not swap_cond
  --
  r2 = (side + sqrt(float((side-1)*(side-1)*.5))*cos(anglez))*.5
  r2 = r2*(2-activity_slide)*.5
  destRect.updatez(r2, anglez, side-1)
  srcImg = roto
  srcRect = srcImg.rect
  offscreenBuffer.copyPixels(srcImg, destRect.rekt, srcRect, \
                             [#blendLevel:blendlev, #ink:inklev, #color:kolor.kolor])
  --
  (the stage).image.copyPixels(offscreenBuffer, (the stage).image.rect, offscreenBuffer.rect, \
                               [#blendLevel:blendlev, #ink:inklev, #color:kolor.kolor])
  (the stage).image.copyPixels(mzk_img, (the stage).image.rect, mzk_img.rect, \
                               [#ink:36])
  --
  old_bin = bin_string
end

--------
--mzk
property offscreenBuffer, side

on new me, s
  side = s
  offscreenBuffer = image(side, side, 1)
  offscreenBuffer.fill(offscreenBuffer.rect, rgb(0,0,0))
  offscreenBuffer.fill(offscreenBuffer.rect, [#shapeType: #oval, #color: rgb(255,255,255)])
  return offscreenBuffer
end
--------
--roto
property offscreenBuffer, side, circles

on new me, s, zcale
  side = s
  circles = [ new(script "circle", 0, 0, 150),   \
              new(script "circle", -37, 0, 105), \
              new(script "circle", 2, 0, 60),    \
              new(script "circle", -13, 0, 40),  \
              new(script "circle", 3, 0, 20)]
  offscreenBuffer = image(side, side, 1)
  repeat with i = 1 to circles.count()
    cx = circles[i].cx*zcale
    cy = circles[i].cy*zcale
    r = circles[i].r*zcale
    lx = side*.5+cx-r
    ly = side*.5+cy-r
    o_rect = rect(lx, ly, lx+2*r, ly+2*r)
    offscreenBuffer.draw(o_rect, [#shapeType: #oval, #lineSize: 2, #color: rgb(0,0,0)])
  end repeat
  return offscreenBuffer
end
```

```
--img_feedback
property offscreenBuffer, destRect, srcRect
property side
property suplefactor, blendlev, inklev
property activity, friction
property r_off, angle_off, d_angle_off, off_rect
property bin_string
--
on new me, s
  side = s
  offscreenBuffer = image(side, side, 24)
  offscreenBuffer.fill(offscreenBuffer.rect, rgb(0,0,0))
  --
  suplefactor = -8
  blendlev = 255
  inklev = 0
  --
  friction = .01
  angle_off = 0
  r_off = 8
  return me
end

on updatez me, bin
  bin_string = bin
  activity =  0
  repeat with i=1 to 8
    if char i of bin_string = 1 then
      activity = activity + 1
    end if
  end repeat
  suplefactor = suplefactor + (-8-activity*.4-suplefactor)*friction
  angle_off = angle_off + PI/(12-activity*.5)
  dr = (6-activity*.5) - r_off
  r_off = r_off+dr*friction
  off_rect = rect(r_off*cos(angle_off), r_off*sin(angle_off), r_off*cos(angle_off),
r_off*sin(angle_off))
  srcRect = rect(0,0,side-1,side-1)+off_rect
  destRect = [point(0, suplefactor), point(side-1-suplefactor,0), \
              point(side-1,side-1-suplefactor), point(suplefactor,side-1)]
  offscreenBuffer.copyPixels((the stage).image, destRect, srcRect, \
                             [#blendLevel:blendlev, #ink:inklev])
  --
  (the stage).image.fill((the stage).image.rect, rgb(0, 0, 0))
  (the stage).image.copyPixels(offscreenBuffer, (the stage).image.rect, offscreenBuffer.rect, \
                               [#blendLevel:blendlev, #ink:36])
end
--------
--kolor_cycling
property palettez, indez, kolor

on new me, i
  palettez = [rgb(255, 0, 0), rgb(0, 255, 0), rgb(0, 0, 255), rgb(0, 0, 0)]
  indez = i
  return me
end

on cycle me, direktion
  if(direktion<0) then
    indez = indez - 1
    if indez < 1 then
      indez = palettez.count()
    end if
  else if(direktion>0) then
    indez = indez + 1
    if indez > palettez.count() then
      indez = 1
    end if
  end if
  kolor = palettez[indez]
end
--------
--kuad
property rekt

on new me
  return me
end

on updatez me, r, a, side
  rekt = []
  repeat with i=1 to 4
    ang = (3/4 + i*.5)*PI + a
    rekt[i] = point(r*cos(ang)+side*.5, r*sin(ang)+side*.5)
  end repeat
end
----------
beam_zone
on new me, s
  zones = []
  srcImg = new(script "beam_set", s, 66)
  blendlev = 255
  inklev = 33
  k = 1
  repeat with i= 0 to 1
    repeat with j= 0 to 1
      zones[k] = image(s, s, 1)
      zones[k].fill(zones[k].rect, [#shapeType: #rect, #color: rgb(0,0,0)])
      srcRect = rect(i*s*.5, j*s*.5, (i+1)*s*.5, (j+1)*s*.5)
      zones[k].copyPixels(srcImg, srcRect, srcRect, \
                          [#blendLevel:blendlev, #ink:inklev])
      k = k + 1
    end repeat
  end repeat
  return zones
end
--------
```

```
--beam_set
property offscreenBuffer
property side
property srcImg, srcRect, destRect, blendlev, inklev, kolor

on new me, s, beams
  side = s
  offscreenBuffer = image(side, side, 1)
  srcImg = member("pointer").image
  srcRect = srcImg.rect
  blendlev = 255
  inklev = 33
  offscreenBuffer.fill(offscreenBuffer.rect, [#shapeType: #rect, #color: rgb(0,0,0)])
  if beams <> void then
    me.beam(beams)
  end if
  return offscreenBuffer
end

on beam me, beams
  repeat with i= 0 to beams-1
    a = dec2rad(i*360/beams-1)
    r = 15*side/16
    me.displayPointer(a)
  end repeat
end

on displayPointer me, a
  r = sqrt(float((side-1)*(side-1)/2))
  destRect = new(script "kuad")
  destRect.updatez(r, a, side)
  offscreenBuffer.copyPixels(srcImg, destRect.rekt, srcRect, \
                             [#blendLevel:blendlev, #ink:inklev])
  destRect = void
end
--------
--zoniko
property side
property fadeTimer, minAmbVol, maxAmbVol
property zaund_array
property vol_level, vol_adjust
property regularChannels, pitchChannel, nextPitchChannel, bgChannel
property a, r
property old_bin_string
--
on new me, s
  side = s
  minAmbVol = 64
  maxAmbVol = 255
  fadeTimer = 1000
  vol_level = 1
  vol_adjust = 1
  zaund_array = []
  repeat with i = 1 to 14
    zaund_array[i] = member i of castLib "znd"
  end repeat
  repeat with i=1 to 4
    sound(i).play([#member: member(zaund_array[i+2]), #rateShift: 0])
  end repeat
  bgChannel = 6
  sound(bgChannel).play([#member:member(zaund_array[2]), #rateShift: 0])
  pitchChannel = 7
  nextPitchChannel = 8
  me.pitchFake(0)
  return me
end

on updatez me, feedback
  a = feedback.angle_off
  r = feedback.r_off
  if(feedback.bin_string <> old_bin_string) then
    repeat with i= 1 to 4
      z1 = char i of feedback.bin_string=1
      z2 = char (i+4) of feedback.bin_string=1
      if ((z1 and not z2)  or (z2 and not z1)) then
        c_zaund = zaund_array[i+2+4]
        aux = 0
      else
        if (z1 and z2) then
          c_zaund = zaund_array[i+2+8]
          aux = 1
        else
          c_zaund = zaund_array[i+2]
          aux = 2
        end if
      end if
      sound(i).play([#member:member(c_zaund), #rateShift: 0])
    end repeat
    me.pitchFake(-(8-feedback.activity)*2)
  else
    repeat with i= 1 to 4
      ang = -(a+(i*3+aux)*(360/12))
      sound(i).pan = 255*cos(ang)
      sound(i).volume =  minAmbVol + (maxAmbVol-minAmbVol)*sin(ang)
    end repeat
    sound(bgChannel).pan = 255*sin(-a)
    sound(pitchChannel).pan = 255*cos(a)
    sound(pitchChannel).volume = minAmbVol + (maxAmbVol-minAmbVol)*sin(a)
  end if
  old_bin_string = feedback.bin_string
end

on pitchFake me, pitch
  sound(nextPitchChannel).play([#member: member(zaund_array[1]), #rateShift:
pitch])
  tmp = pitchChannel
  pitchChannel = nextPitchChannel
  nextPitchChannel = tmp
end
```







La tecnologización es un proceso al que nuestras sociedades se han ido integrando de forma tardía, respecto de las sociedades productoras de tecnología.

Consecuentemente con el desarrollo de la tecnología digital y su volcamiento sobre el mercado, se ha llegado a una equiparación en cuanto a los objetos tecnológicos a los que se tiene acceso; el mercado local se abre al mercado global.

De esta forma, el medio local presenta un panorama tecnológico diverso (en la medida en que la modernización de las estructuras es paulatina, más tardía, posterior y no congruente) donde se traslapan objetos/sistemas altamente tecnologizados con un sistema precario. Esta hibridación tecnológica produce descalces e ineficiencias que se traducen en pérdida.

En este contexto, el acceso al objeto tecnológico actúa como medio de representacion social, a la vez que la tecnologización, como proceso, es asumida como estrategia de integración del medio local en el contexto globalizado.

Más allá de su hipercapacidad medial y de simulación, el poder de los nuevos medios reside en ser un modelo de interpenetración en y sobre lo real que nos pone en una situación extática y marginal respecto de sus propios procesos. La reubicación a modo de terminales en la que somos dispuestos manifiesta una violencia silente que modifica la experienciación de nuestros cuerpos en tanto entidades informadas.

La relación cuerpo/información puede ser entendida como una operación donde ambos parámetros son excluyentes y, por tanto, requeribles de mediar. Dicha relación sólo puede establecerse en tanto el primero sea informable, es decir, mediante procesos de corte/registro como operaciones diferenciales de decodificación/codificación.

A su vez, la relación información/cuerpo funciona sólo en campos de acción dispersa y asincrónica mediante su representación/reespacialización. Teniendo como base que la información es una reducción de flujo desde un sistema contínuo a uno discreto, ésta requiere de ser amplificada y desinformada mediante sistemas de representación consensuales.

Las amplificaciones de señal -o extensiones- producidas por los media, condicionan su eficacia/credibilidad al tiempo de respuesta y a la predictibilidad de los resultados esperados, desatendiendo fragmentos de código aislados no decodificables que puedan producir fricciones entre sistema y entorno, y evitando descalces entre fuentes emisoras y receptoras que puedan ser causa de interferencia o ruido.

Siguiendo una lógica similar, en cuanto al acceso acrítico (y repentino) a las nuevas tecnologías, y dada la carencia de herramientas técnicas y conceptuales adecuadas, la producción artística local se ha acercado a ellas casi exclusivamente como un desplazamiento del soporte de obra, ajena a las posibilidad de una exploración detenida de los lenguajes, desde la modificación/intervención de los dispositivos preexistentes a la construcción de dispositivos/soportes propios de distribución congruentes con su entorno de producción.