/* Evaluation of preservation strategies for an interactive, software-based artwork with complex behavior */

- Claudia Roeck, University of Amsterdam (UvA),
- Julia Noordegraaf, UvA,
- Klaus Rechert, University of Freiburg (Germany)
• Introduction / Research Question;
• Method and Case Study;
• Significant Properties;
• Preservation Strategies;
• Conclusions;
Introduction / Research Question;
Complexity of Digital Artworks

One-component digital object, standard conditions

Ecosystem of objects, non-standard
Literature about Digital Preservation Strategies

- Emulation workflow
- Scaling of emulation
- Emulation of video games, research data, software-based art
- Etc.

- Migration in business: minimal interruption.
- Migration in libraries: format migrations.
Which preservation strategy for software-based art is most sustainable in the long-term?

• What preservation strategies are possible?

• Are there combinations and in-betweens of the well-known preservation strategies?

• What impact do these preservation strategies have on the authenticity of the work?
Method and Case Study;
Horizons (2008) by Geert Mul

https://geertmul.nl/projects/horizons/
Shan Shui (2013) by Geert Mul. Video by Jose Biscaya and Rachel Sommers, LIMA.
Artwork Analysis;
**Digital preservation Strategies for Horizons**

<table>
<thead>
<tr>
<th>Reprogramming</th>
<th>Migration</th>
<th>Virtualization</th>
<th>Emulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translation to different programming language</td>
<td>Transfer of program to new or changed software environment → Small adaptations, recompilation</td>
<td>Replaces the computer (but not the CPU) by a software layer. Direct access to CPU.</td>
<td>Replaces the whole computer by a software layer.</td>
</tr>
<tr>
<td>Conceptually Migration / Changing object</td>
<td>Encapsulation / Freezing object</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Direct access to CPU.
- Replaces the whole computer by a software layer.
Adaptability to new hardware.
Resilience to software obsolescence and adaptability to changed network protocols.

Stabilization of software complexity and minimizing change rate.
Scalability of preservation strategy.
Ease of installation of the artwork / of connecting peripherals.

Criteria for sustainability of preservation strategies

Ease of maintenance / execution of function tests.
Resilience to software obsolescence and adaptability to changed network protocols.
Significant properties of Horizons;
**Significant Properties: what to preserve?**

**What to preserve?**
- Behaviour / interactivity
  - Generative video and sound
  - Quality of video movement (speed, smoothness, pattern) and of sound

**What to consider?**
- Technical dependencies
  - Tracking sensor (hard coded in source code)
  - Video card (hard coded in source code)
Runtime Environment and Source Code of *Horizons*

- **Runtime Environment**
  - Images
  - Ruby Libraries
  - C Libraries

- **Runtime Environment**
  - Ruby Interpreter

- **Ruby-Programs**
  - C-Program (shared object file)

- **C-Compiler**
  - C-Source Code

- Script not used (remains)
What to preserve?

Behaviour / interactivity

Generative video and sound

Quality of video movement (speed, smoothness, pattern) and of sound

What to consider?

Technical dependencies

Tracking sensor (hard coded in source code)

Video card (hard coded in source code)

replaceable

replaceable

Complete description not possible
Significant Properties;

What to preserve?

- Behaviour / interactivity
  - Generative video and sound
  - Quality of video movement (speed, smoothness, pattern) and of sound

What to consider?

- Technical dependencies
  - Source Code (replaceable)
  - Tracking sensor (hard coded in source code)
  - Video card (hard coded in source code)

Complete description not possible
Preservation Strategies Applied;
<table>
<thead>
<tr>
<th>Reprogramming</th>
<th>Migration</th>
<th>Virtualization</th>
<th>Emulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Translation to different</td>
<td>Transfer of program to new or changed</td>
<td>Replaces the computer (but not the CPU) by</td>
<td>Replaces the whole computer by a software</td>
</tr>
<tr>
<td>programming language</td>
<td>software environment → Small adaptations,</td>
<td>a software layer. Direct access to CPU.</td>
<td>layer.</td>
</tr>
<tr>
<td></td>
<td>recompilation</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Conceptually Migration / Changing</td>
<td></td>
<td>Encapsulation / Freezing object</td>
<td></td>
</tr>
<tr>
<td>object</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Migration

Debian 7
Source code
Specific video card

Debian 9
Adapted and recompiled Source code
Any video card

Stabilization of software complexity | Hardware independency | Performance / behaviour
--- | --- | ---
Middle. | Middle | Good. (direct access of CPU and video card)
Future adaptations of source code probable | Disk image transferable to same CPU architecture. | Video pattern slightly different

Source code
Adapted and recompiled Source code
Stabilization of software complexity
Hardware independency
Performance / behaviour
Specific video card
Generalization of video card
Any video card
Hardware independency
Performance / behaviour
Middle
Middle
Good

Debian 7
Source code
Specific video card

Debian 9
Adapted and recompiled Source code
Any video card

Stabilization of software complexity | Hardware independency | Performance / behaviour
--- | --- | ---
Middle. | Middle | Good. (direct access of CPU and video card)
Future adaptations of source code probable | Disk image transferable to same CPU architecture. | Video pattern slightly different

Source code
Adapted and recompiled Source code
Stabilization of software complexity
Hardware independency
Performance / behaviour
Specific video card
Generalization of video card
Any video card
Hardware independency
Performance / behaviour
Middle
Middle
Good
“Original” (version 2013)
Debian 7, artist’s computer

Migration
Debian 9, artist’s computer
Hardware abstraction / graphics chain of *Horizons*;
Virtualization and Emulation;

<table>
<thead>
<tr>
<th>Method</th>
<th>Stabilization of software complexity</th>
<th>Hardware independency</th>
<th>Performance / behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Emulation</td>
<td>Good</td>
<td>Good</td>
<td>Too slow (no direct access of CPU and video card)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disk image / Snapshot transferable to any computer hardware</td>
<td></td>
</tr>
<tr>
<td>Virtualization</td>
<td>Good</td>
<td>Middle</td>
<td>Too slow (direct access of CPU)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Disk image / Snapshot transferable to same CPU architecture</td>
<td></td>
</tr>
</tbody>
</table>
Paravirtualization;

- Migrated version of Horizons
- Any video card

Virtualization of CPU and video card

- Migrated version of Horizons. Generalized video card driver
- Any video card

<table>
<thead>
<tr>
<th>Stabilization of software complexity</th>
<th>Hardware independency</th>
<th>Performance / behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>Middle.</td>
<td>Middle</td>
<td>Good for video, bad for sound.</td>
</tr>
<tr>
<td>Adaptation of guest OS-kernel to underlying host hardware necessary</td>
<td>Disk image / Snapshot transferable to same CPU Architecture. Any video card with OpenGL</td>
<td>(direct access of CPU and video card)</td>
</tr>
</tbody>
</table>
Virtualization with video card pass-through:

- Migrated version of Horizons
- Any video card
- Specific video card

Virtualization of CPU
- Pass-through video card
- Migrated version of Horizons. Driver of specific video card.

Stabilization of software complexity
- Good

Hardware independency
- Bad
- Disk image / Snapshot Video card and CPU dependent

Performance / behaviour
- Good (direct access of CPU and video card)

Not tested
Tracking sensor in virtualization / emulation;

<table>
<thead>
<tr>
<th>Sensor handling in an emulation / virtualisation of Horizons</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Guest</strong></td>
</tr>
<tr>
<td>Artifact</td>
</tr>
<tr>
<td>sick.c (driver sick sensor)</td>
</tr>
<tr>
<td>USB driver</td>
</tr>
<tr>
<td>Emulation / Virtualisation of CPU</td>
</tr>
<tr>
<td>sick sensor</td>
</tr>
<tr>
<td>USB pass-through</td>
</tr>
</tbody>
</table>
Conclusions;

• No clear winner of preservation strategies in the mid-term for this work.

• In the long-term, the encapsulation of the work clearly separates it from additions (interface for sensor) in the guest system and reduces its change rate (video pattern). These are both a principles in conservation.

• Paravirtualization: good solution in the mid-term and serves as a transition to / preparation for emulation in the future

• Paravirtualization: combination and merging of emulation and migration
Thanks to

- Julia Noordegraaf, University of Amsterdam
- Klaus Rechert, University of Freiburg
- Gaby Wijers, LIMA, Amsterdam (www.li-ma.nl)
- Geert Mul, Rotterdam
- Teal Gaure, Amsterdam
Thank you;
c.rock@uva.nl