Games Engineering

Wissenschaft mit, über und für Interaktive Systeme

SEBASTIAN VON MAMMEN, ANDREAS KNOTE, DANIEL ROTH,

MARC ERICH LATOSCHIK

EINLEITUNG

Video- und Computerspiele sind im 21. Jahrhundert zu einem relevanten Kulturgut der digitalen Ära und zu einem wirtschaftlich wichtigen Faktor der Unterhaltungsindustrie geworden. Sie haben einen maßgeblichen Beitrag zur Verbreitung und Marktdurchdringung von Computersystemen im privaten Bereich geleistet (Gallagher et al., 2002). Computerspiele genießen einen hohen Stellenwert im Vergleich zu anderen Softwareprodukten und verzeichnen hohe Absatzzahlen seit Einführung der sogenannten Personalcomputer (Hertz, 1997) bis heute (Bundesverband Interaktive Unterhaltungssoftware, 2017).

Computerspiele dienen oft als Messlatte für die Leistungsfähigkeit heutiger Computersysteme. Viele *Triple-A-Titel*[1] orientieren die benutzten technischen Verfahren werbewirksam am aktuellen Stand der Technik oder definieren diesen sogar neu, um sich so im Markt besser zu positionieren und Vorteile gegenüber konkurrierenden Produkten zu erlangen. Dadurch

[1] Spieletitel, welche aufwändig mit großem Investitionsvolumen und weltweiter Verbreitung, meistens von den großen Firmen der Branche, entwickelt werden und in der Regel hohe Absatzzahlen verbuchen (Triple-A oder AAA).

steigt das Interesse an der damit einhergehenden technischen Expertise, den notwendigen Fachkenntnissen, diese Systeme auszureizen - siehe bspw. die seit Anfang der 2000er Jahre erscheinende *Game Programming Gems* Buchserie von Rivermedia.

Die Nutzung von Computersystemen zur Entwicklung spielerischer Inhalte hat eine weitreichende Historie, die fast zeitgleich mit der Computerentwicklung begonnen hat und mindestens bis in die 50er Jahre des 20. Jahrhunderts zurückreicht (s. *OXO*, *Tennis for Two oder Spacewar!)*. Von Anfang an mussten Spieleentwickler eine Vielzahl informatischer Technologien bedienen, um ein erfolgreiches Spiel oder später Produkt zu realisieren. Eingaben der Spielerinnen und Spieler[2] über die verschiedenen Eingabegeräte müssen verarbeitet werden (siehe Roth et al., 2017), den Spielzustand gemäß der umgesetzten Spielemechaniken verändern und als Konsequenz das entsprechende, in der Regel visuelle und auditive Feedback, initiieren. Eine korrekte Umsetzung dieser Interaktionsschleife ist eine der Voraussetzungen, um das Spielvergnügen zu maximieren und die User Experience (Ux) zu optimieren (Preim & Dachselt, 2015). Das bedingt natürlich nicht nur ein entsprechend ausgeklügeltes, menschzentriertes Design, sondern es motiviert auch dazu, technische Lösungen zu finden, um Eingaben und Ausgaben konsistent und fließend, d.h., ohne wahrnehmbare Brüche und Verzögerungen (so genannte Latenzen) ineinandergreifen zu lassen.
Computerspiele sind interaktive Systeme. Sie stellen je nach Spielgenre besondere nicht-funktionale, speziell zeitliche Anforderungen an die Umsetzung des Prinzips der Eingabe-Verarbeitung-Ausgabe (EVA). Betrachtet man funktionalen Anforderungen zur Erstellung von Computerspielen genauer, kann man verschiedene Aspekte unterscheiden, die in quasi Echtzeit3 umgesetzt werden müssen. Nutzereingaben müssen je nach System und Spiel über eine Vielzahl von möglichen Eingabegeräten erfasst und interpretiert werden. Dies beinhaltet Standardeingabegeräte wie etwa Tastaturen, Mäuse, Trackballs, Touch-Pads und Touch-Displays ebenso wie dedi-

[2] In der Folge werden wir häufig auf die geschlechtsspezifische Konkretisierung aus Gründen des Leseflusses verzichten und beziehen uns damit sowohl auf weibliche Spielerinnen als auch männliche Spieler.
[3] Die Ausprägung der Echtzeitanforderungen (weich, fest, hart) richtet sich unter anderem nach dem Spielgenre, dem Spielprinzip sowie den Ein-/Ausgabemetaphern, z.B. der Nutzung von VR-Systemen.

zierte Geräte speziell für Computerspiele, z.B. spezielle Buttons, Joypads, Joysticks, 3D-Eingabecontroller oder sogar explizite berührungsfreie Kamera-basierte Systeme. Der über die Eingaben beeinflusste interne Spielzustand wird ebenfalls von einer Reihe von funktionalen Anforderungen determiniert. Von den Inhalten, die dem Spieler vermittelt werden, erwartet man Dynamik, also dass sie sich im Spielverlauf verändern oder weiterentwickeln. Diese Dynamik setzt beispielsweise Möglichkeiten voraus, Bilder (*sprites*) oder zusammengesetzte, dreidimensionale Körper (*rigged bodies*) zu animieren, virtuelle Gegenstände physikalisch miteinander wechselwirken oder Nicht-Spieler-Charakteren (*non-player characters, NPCs*) sich intelligent verhalten zu lassen (Künstliche Intelligenz). Alle diese Funktionen müssen im Sinne einer konsistenten, glaubwürdigen und unterhaltsamen Simulation genutzt und kombiniert werden. Am Ende steht die Erzeugung visueller, hörbarer und auch fühlbarer Eindrücke und deren Ausgabe auf Bildschirmen, Lautsprechern oder den Controllern (die hiermit zu kombinierten Ein-/Ausgabegeräten werden). Daneben gibt es noch eine Vielzahl weiterer technischer Aspekte, die für das Funktionieren moderner Videospiele grundlegend sind, beispielsweise für die Kommunikation mehrerer Spiele(r) in einem Netzwerkverband oder über das Internet, die Speicherung von Spielständen lokal oder in der Cloud, etc.

Über lange Zeit hinweg mussten Spieleentwickler die notwendigen informatischen Funktionalitäten in der Regel von Grund auf selbständig implementieren. Typische Beispiele für solche Funktionalitäten sind die Verwaltung von Speicherblöcken (oder Ressourcen allgemein) für das Vorhalten größerer Datenstrukturen, Hardware-spezifische Algorithmen für das Abspielen von Tönen und Musikdateien über die verfügbaren Soundkarten oder eigene Graphikroutinen zur Generierung der visuellen Feedbacks.

Auch heute haben low-level Implementierungen für spezielle Funktionen ihren Stellenwert, etwa im Bereich eigens entwickelter Spielkonsolen oder bei einem allgemeinen Fehlen entsprechender Betriebssoftware bzw. *Middleware* (*Engines,* s.u.). Allerdings haben die Verbreitung der Computerspiele an sich, deren steigende Komplexitäten der Codebasen und Spielewelten Trends zur Standardisierung und Modularisierung verstärkt. Software-Bibliotheken, die standardisierte Designs und Befehlsschätze für bestimmte Funktionalitäten zur Verfügung stellen, wurden an die eigene Codebasis angebunden. Sofern eine dieser Software-Bibliotheken verwendet wird, um Inhalte eines Videospiels kontinuierlich zu verändern, spricht

man auch von sogenannten *Engines* (englisch für Motoren) (Lewis & Jacobson, 2002). Wie der Motor eines Autos die Räder, so treiben Softwareengines die Berechnung der notwendigen etwa graphischen Darstellung virtueller Objekte oder ihrer physikalischen Zustände an. Dedizierte Bibliotheken und Engines werden heute häufig als open-source Projekte entwickelt, über das Internet verbreitet und dadurch leicht zugänglich gemacht. Auch kommerzielle Anbieter gehen heute diesen Weg und Partizipien im Sinne der wirtschaftlichen Erträge erst über mögliche spätere Spielumsätze. Vollständige Game Engines nehmen dem Entwickler zusätzlich die Arbeit ab, verschiedene spezifische Engines zu integrieren und geben ihm möglichst einfache Rezepte an die Hand, wie man Daten über virtuelle Welten, deren Bewohner und ihre Verhalten verwaltet und komponiert, und den Spieler wie gewünscht damit interagieren lassen kann.
Dieser Anspruch von Game Engines, verschiedene funktionale Aspekte effizient und zugänglich für die Schaffung und Ausgestaltung interaktiver Systeme zu integrieren, motiviert die Entstehung des neuen Forschungsgebiets Games Engineering. Games Engineering fokussiert sich auf Entwurf, Entwicklung und Verbesserung von Algorithmen, Datenstrukturen, Code Modulen, Entwurfsmustern und Software Engineering Prinzipien rund um Engines, Plugins, Testverfahren und zugehörige Tools für computerbasierte Spiele. Das Repertoire für Games Engineering Technologien reicht von hardwarenaher Optimierung über intuitive Bedienungsschnittstellen bis zur technologiegestützten Erstellung konkreter Inhalte virtueller Welten.

Das Ziel der Systementwicklung im Bereich Games Engineering ist stets, interaktive Systeme besser und schneller ausgestalten zu können. Sprach man in den 1960er Jahren noch von *On-Line Systemen* (Jones, 1967), die auf Benutzereingaben hin verschiedene Simulationsverläufe präsentierten, spricht man heutzutage allgemein von *Real-time Interactive Systems* (RIS), oder *Echtzeit-interaktiven Systemen* (Wiebusch und Latoschik, 2015). Vermeintlich lässt sich die terminologische Entwicklung auf die Verknüpfung mit Anwendungsdomänen zurückführen. Betrachtet man RIS mit starkem Simulationsbezug bewegt man sich im Bereich der *Human-in-the-Loop Simulationen* (Narayanan und Rothrock, 2011). Am Beispiel von Crysis 3 (2013, EA), einem technisch anspruchsvollen, actionreichen First-Person-Shooter, kann man leicht nachvollziehen, wie ein Videospiel als RIS verstanden werden kann: die Benutzereingaben müssen schnell und ohne Verzögerung verarbeitet werden. Die unterschiedlichen Berechnungen

- Grafik, Physik, KI - müssen mit der jeweils notwendigen Frequenz berechnet werden, ohne sich gegenseitig zu blockieren oder zu verzögern. Die Datenverarbeitung unterliegt festen Echtzeit-Bedingungen (*firm realtime constraints*), da verspätete Reaktion oder Wahrnehmungen das Spielerlebnis deutlich beeinträchtigen. Es ist die Identifikation gemeinsamer funktionaler und nicht-funktionaler Anforderungen, welche die Brücke schlägt zwischen Computerspielen und der Entstehungsdomäne für RIS: Anwendungen im Bereich der *Virtual Reality (VR)* und *Augmented Reality (AR)*. Die Verfügbarkeit bezahlbarer VR-Geräte im *Consumer*-Markt macht diese Nähe heute zusehends explizit. In Computerspielen wie in VR/AR-Anwendungen geht es um interaktiv erlebbare Inhalte unter Einhaltung bestimmter sehr ähnlicher funktionaler und nicht-funktionaler Randbedingungen der Konsistenz und Zeitlichkeit. Unterschiede bestehen, wenn, dann eher im Anwendungsziel, also der Frage, ob es bei der Anwendung primär um die Unterhaltung der Benutzer geht oder ob diese einen im weitesten Sinne produktiven Anspruch hat. Allerdings verschwimmen auch hier zusehends die Grenzen, siehe dazu die Nutzung von RIS-Technologien für *Serious Games* (Charsky, 2010) oder digitale interaktiver Kunstinstallationen (Paul, 2003). Durch die große konzeptionelle und technische Nähe und den kommerziellen Erfolg von Computerspielen haben sich Game Engines heute als weit verbreitete Plattformen zur Entwicklung von RIS etabliert (bspw. Steed, 2008). Diese Entwicklung ist zum einen begrüßenswert, da sich mit Hilfe dieser Engines ein beeindruckender Funktionsumfang heutiger RIS-Anwendungen leichter realisieren lässt und sich VR/AR-Anwendungen einem größeren Publikum erschließen. Zum anderen besteht hier aber auch eine gewisse Sensibilität, da die Anforderungen an VR/AR-Systeme bezüglich der nicht-funktionalen Eigenschaften im Gegensatz zu vielen nicht-VR-basierten Computerspielen deutlich verschärft sind, um Risiken möglicher Nebenwirkungen (etwa Cyber Sickness) zu minimieren.

In diesem Kapitel haben wir beleuchtet, welche wissenschaftlichen Meilensteine und Strömungen im Games Engineering kulminieren, welche exemplarischen Querverbindungen zu anderen Wissenschaften und anderen Disziplinen der Informatik bestehen und welche aktuellen Trends sich im Games Engineering abzeichnen. Im Folgenden widmen wir uns der Eingrenzung des Forschungsgebiets Games Engineering insbesondere auch in Bezug auf relevante Studiengänge und die wissenschaftliche Community. Im Anschluss betrachten wir marktwirtschaftliche Vorgänge, die Games

Engineering als integrative Disziplin angeschoben haben, bevor wir Kernkomponenten erläutern, also die verschiedenen funktionalen Aspekte, die in Game Engines üblicherweise zusammengeführt werden. Es gibt eine Reihe von Softwaretechnologien, die diese Zusammenführung unterstützen oder die konkret im Kontext entwickelt wurden, z. B. Szenengraphen, Eventsysteme oder das Entity-Component-System Entwurfsmuster. Schließlich stellen wir noch Bezüge des Games Engineering zu anderen Wissenschaften her, bevor wir dieses Kapitel kurz zusammenfassen.

EINGRENZUNG DES FORSCHUNGSGEBIETES

Das Forschungsgebiet Games Engineering befindet sich gegenwärtig (2018) immer noch in der Ausdifferenzierung. Gleichzeitig nutzt das Forschungsgebiet bereits eine Vielzahl wichtiger vorausgehender Forschungsarbeiten speziell aus der Informatik. Games Engineering und Software Engineering ähneln sich schon begrifflich. Games Engineering kombiniert Software Engineering speziell mit Games-spezifischen informatischen Anwendungsgebieten der Computergraphik, physikalischen Simulation, Künstlichen Intelligenz, Virtuellen Realität und Mensch-Computer-Interaktion im Allgemeinen. Daraus ergibt dich eine Definition wie folgt:

> *Games Engineering beschäftigt sich mit der zielorientierten Bereitstellung und systematischen Verwendung von Prinzipien, Methoden und Werkzeugen für die arbeitsteilige, ingenieurmäßige Entwicklung und Anwendung von umfangreichen Softwaresystemen für Computerspiele.*

Mit dieser Definition differenziert sich Games Engineering von verwandten Gebieten und Arbeitsbereichen in der Games-Branche und in verwandten Industrien, speziell Game Design, Game Development und Game Informatics. Games Engineering fokussiert sich auf die Schaffung und Weiterentwicklung wissenschaftlicher Ansätze und technologischer Lösungen für das softwaretechnische Design und die Entwicklung interaktiver Systeme. Game Design beschreibt die Fähigkeiten, primär inhaltliche und gestalterische Konzepte für Videospiele zu entwickeln, den Spieler in Relation zur Spielewelt zu setzen und mit ihr durch Regeln und Spielemechanismen interagieren zu lassen (Pedersen, 2003). Game Development hingegen umfasst die üblicherweise agile, durch Playtesting geprägte Entwicklung von Videospielen (Bethke, 2003). Game Informatics wiederum schlägt die Brücke zwischen Game Design und Game Development.

Games Engineering Studiengänge

Games Engineering stellt einen Forschungsbereich dar, der die für Games und interaktive Systeme im allgemeinen grundlegenden Technologien zum Gegenstand hat. Daraus ergeben sich gleichnamige oder namensähnliche Studiengänge, für die man sich an verschiedenen deutschen Hochschulen und Universitäten einschreiben kann - Games Engineering, Informatik: Games Engineering, Informatik oder Software Engineering mit Schwer-

punkt Games Engineering. Neben informatischen Grundlagen, sowie Grundzügen verwandter Gebiete wie des Game Designs und extensiven Erfahrungen im Game Development vermitteln diese Studiengänge insbesondere Wissen und Fähigkeiten, um neue Engines zu entwickeln, tief in existierende Engines einzutauchen, sie zu analysieren, ihre grundsätzlichen Konzepte nachzuvollziehen, sie zu erweitern oder zu optimieren und durch notwendige Werkzeuge zu unterstützen. Die unterschiedlichen Zielsetzungen von Engines werden dabei durch verschiedene Methoden unterstützt - auf Ebene der hardwarenahen Implementierung, der Algorithmen, der Softwarearchitektur und deren Schnittstellen. Hinzu kommen Methoden des Profiling bzw. der Performanzmessung von echtzeitfähigen Routinen (Coppa et al., 2014) sowie der Usability und User Experience (Preim & Dachselt, 2015). Weiterführende Methoden und Fragestellungen ergeben sich aus den konkreten Zielsetzungen von Engines, die maßgeblich für die Generierung, Simulation oder Verwaltung von Inhalten relevant sind, wie beispielsweise der Evaluation photorealistischer Renderings durch physikalisch-basierte Computer Grafik Engines (Pharr et al., 2016) oder der Beurteilung rationalen Verhaltens von NPCs gesteuert durch Engines für künstliche Intelligenz (Bellemare et al., 2015).

Games Engineering Community

Es gibt eine Vielzahl von Forschergruppen, die sich mit Fragestellungen des Games Engineering auseinandersetzen. Üblicherweise fokussieren sie sich auf spezielle Fragestellungen oder Teilaspekte, wie beispielsweise künstliche Intelligenz in Videospielen. *Serious Games*, also Videospiele mit ernsthaftem Hintergrund (Charsky, 2010), definieren dabei meist die primäre Anwendungsdomäne, und deren Effektivität und Spielspaß motivieren die Bearbeitung weiterführender technologischer Fragestellungen. Der steigende Bedarf, technologische Lösungen zu erforschen – für eine wachsende Anzahl an Entwicklern, Designern, Spielern und Anwendern – spiegelt sich nicht nur in der Vielzahl neuer Forschergruppen wider, die sich auf Games Engineering fokussieren. Auch steigt die Anzahl und der Einfluss kleiner, mittelständischer und großer Unternehmen, die Games Engineering Produkte innovieren und auf den Markt bringen.

Entsprechend vermischen sich auch industrielle und wissenschaftliche Plattformen des Wissensaustauschs. Jährliche Messen wie Gamescom oder GDC (Game Developer Conference) sprechen primär Wirtschaftsvertreter

und Konsumenten an, selbst wenn dort einschlägige Präsentationen und Workshops zu aktuellen wissenschaftlichen Themenbereichen angeboten werden. Weiterhin gibt es themenspezifische kommerzielle Konferenzen mit dem Anspruch, den State-of-the-Art zu kommunizieren, ohne als Plattform für wissenschaftlichen Diskurs zu dienen, bspw. die Serious Games Conference (SGC) im Rahmen der CeBit oder die Serious Play Conference. Andererseits werden auch auf den primär wissenschaftlichen Konferenzen Referenten aus der Industrie eingeladen.

Grob einteilen lassen sich die wissenschaftlichen Konferenzen und Journals in die Bereiche Games Engineering und Serious Games Research, letzteres oftmals mit einem Fokus auf Wissensgewinn und Training. Verschiedene Konferenzen setzen in diesem Bereich unterschiedliche Akzente. Die European Conference on Games Based Learning (ECGL) oder die Games+Learning+Society Conference (GLS) stellen das Lernen durch Videospiele in den Vordergrund. Mit den Beitragskategorien Theorie, Technologie, Anwendung, Evaluation bspw. bei der Konferenz zu Virtual Worlds and Games for Serious Applications (VS-GAMES), der Joint Conference on Serious Games (JCSG) oder der Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational & Serious Games (CGAMES), werden auf einigen Konferenzen neben Serious-Games-Aspekten auch Forschungsergebnisse des Games Engineering präsentiert und diskutiert. Die Konferenzen Computational Intelligence in Games (CIG), Artificial Intelligence and Interactive Digital Entertainment (AIIDE), Game-On, Conference on Game Engineering and Artificial Intelligence (ICGEAI), Eurographics und SIGGRAPH wiederum fokussieren sich stärker auf die Schaffung interaktiver, dynamischer Inhalte, was grundsätzlich einen stärkeren technischen Bezug herstellt. Die Exploration neuartiger Interaktionsformen in Videospielen steht bei der Konferenz CHI PLAY im Vordergrund. Auch auf Konferenzen wie der Virtual Reality (VR) oder der Virtual Reality Software and Technology (VRST) werden neuartige RIS-basierte Ansätze für hoch interaktive Mensch-Computer-Interaktion erforscht und diskutiert. Der damit einhergehende generelle Anspruch, echtzeitfähige, interaktive Systeme zu entwickeln, schlägt sich beispielsweise in anhängigen Plattformen wie dem Workshop zu Software Engineering and Architectures for Realtime Interactive Systems (SEARIS) nieder.

FORMATION DURCH WIRTSCHAFTLICHE KRÄFTE

Game Engines sind eine integrative Technologie, die man als Produkt starker marktwirtschaftlicher Kräfte verstehen kann. Der Heimcomputermarkt wurde in den 1980er Jahren auch durch den starken Absatz von Systemen geprägt, welche über die notwendigen Fähigkeiten zur Nutzung als Spielgeräte verfügten und für welche in der Folge umfangreich Computerspiele entwickelt wurden. Systeme wie der Commodore C64, der Amiga oder der Sinclair Spectrum waren durchaus vollwertige Computersysteme auch für ernsthafte Anwendungen, sie verfügten aber ebenfalls früh über teils sehr spezielle Fähigkeiten, die sie als dedizierte Spielsysteme auszeichneten, auch, um sich dadurch Markvorteile gegenüber konkurrierenden Systemen zu sichern.

Seither haben sich Spielekonsolen und PCs stark ausdifferenziert (Finn, 2002). Wie eingangs beschrieben, simulieren finanziell und technisch aufwändige Triple-A-Titel immer imposantere virtuelle Welten, um sich durch beeindruckende Sinneseindrücke Vorreiterrollen auf dem Videospielmarkt zu sichern. Dieser Trend diktiert eine hohe technische Komplexität und bestmögliche Optimierung für die Zielhardware, und somit eine ständige Weiterentwicklung der Kerntechnologien wie Rendering, Physics oder Networking. Die Veröffentlichungszyklen neuer Generationen von Videospielekonsolen haben Entwicklern wie Verbrauchern in den letzten Dekaden ein wenig Planungssicherheit ermöglicht (Schilling, 2003). Der hochgradig facettenreiche PC-Markt entwickelt sich kontinuierlich weiter, ermöglicht dadurch stets maximale Performanz, lässt Standards aber entsprechend schnell veralten. Im Konsolenbereich konnte man bislang hingegen mit fixen Rahmenbedingungen und abschätzbaren Laufzeiten von mehreren Jahren rechnen. Seit Kurzem häufen sich Indikatoren, dass es zu einer Trendwende kommen könnte - Microsoft etabliert für seine XBox Produktserie die Strategie einer einheitlichen Softwareplattform (Universal Windows Platform, UWP) für Spielekonsolen und PCs (Taylor, 2016). Aus technischer Sicht ähneln sowohl Microsofts als auch Sonys PlayStation 4 Spielekonsolen immer stärker handelsüblichen Consumer PCs. Gleichzeitig wurde 2016 mit dem Erscheinen der XBox One S und der PlayStation 4 Pro mit der vermeintlichen Unveränderlichkeit von Konsolenhardware innerhalb einer Generation gebrochen (Webber und Brewster, 2016).

Der hohe technische Anspruch von Triple-A-Titeln bleibt dessen ungeachtet bestehen. Der sich daraus ergebende Wettbewerbsdruck und die damit einhergehende, notwendige technische Innovation spiegeln sich in Analysen von Entwicklungsprozessen und -entscheidungen konkurrierender Spieleentwickler wider. Diese Analysen werfen beispielsweise einen Blick auf die Optimierung der Wertschöpfungsketten und den damit einhergehenden Rollenverteilungen: Ein großer Teil der Arbeit bei der Entwicklung einer Game Engine besteht in der Schaffung einer Toolchain, die es Designern und Künstlern ermöglicht, Inhalte zu generieren ohne von Programmierern abhängig zu sein (Gregory, 2014; Abrash, 1997). Aspekte dieser Toolchains sind beispielsweise die flexible Verwendung etablierter Datenformate oder spezifischer Tools für die Bearbeitung von Spielewelten und -inhalten. Auch warten gängige Games Engines mit umfangreichen Scripting Engines auf, die es ermöglichen, relativ schnell komplexe Spielemechanismen zu implementieren, teilweise sogar unterstützt durch sogenannte visuelle Programmiersprachen, die es gerade für Novizen erleichtern, algorithmische Zusammenhänge zu formulieren (von Mammen et al., 2016).

Genauso wie die Arbeitsteilung bei der Entwicklung von Videospielen stringent und professionell unterstützt und weitgehend eingehalten wird, wird auch dem hohen Entwicklungsaufwand von Game Engines durch Arbeitsteilung und Spezialisierung begegnet. Um sich zu koordinieren und nachhaltig zu arbeiten, münden die Expertenleistungen in Subengines, Codemodulen und Plugins, die in größeren Entwicklungskontexten zum Einsatz kommen (als Teil des „Development Stacks"). Natürlich müssen diese Programme unterschiedliche Ansprüche erfüllen und Märkte bedienen - von großen Gruppierungen mit hoher Expertise und schnellen Arbeitswegen wie in internationalen Gamestudios über Indie-Developer bis zu akademischen Projekten. Entsprechend vielseitig sind auch die Anbieter, die verschiedene Marktnischen besetzen - von Engineering Studios, die ganze Games Engines entwickeln und vermarkten wie Unity Technologies, bis hin zu Middleware Spezialisten wie SpeedTree, die sich auf die prozedurale Generierung virtueller Inhalte fokussieren.

Will man die marktwirtschaftliche Evolution der Rollen im Games Engineering besser verstehen, muss man seinen Blick auch auf die historische Entwicklung von Game Engines werfen (Eberly, 2006). Besonders interessant in diesem Kontext ist die Entstehung moderner Game Engines, die sich aus der Anforderung effizienter Darstellung dreidimensionaler Computer-

grafik ergab. Dabei stechen die Arbeiten des Gamestudios id Software heraus. Ihr First-person Shooter Game „Doom“ ist eines der frühesten Beispiele für eine klare Trennung von Inhalten (Model, Scripting, Animation, Level) und der zugrundeliegenden Engine („idTech 1”, retronym), was den Entwicklungsprozess des Endprodukts bereits leichter zwischen Content Creation durch Designer und Artists sowie der Engine- und Tool Programmierung der Coder aufteilen lässt[4]. Eine Trennung von Content und Engine führt auch dazu, dass ein Großteil der Programmierarbeit nicht mehr „in die Engine”, sondern in die Toolchain fließt (Abrash, 1997). Spätestens mit dem First-Person-Shooter Quake 2 (id Software, 1997) hat id Software sowohl die spielerische Inspiration als auch die technische Grundlage für eine Vielzahl weiterer Titel dieses Genres angeschoben. Letzteres gelang vor allem durch die Lizenzierung ihrer Game Engine „idTech 2”.

Heute werden sehr viele Videospieleprojekte auf Basis lizenzierter Game Engines realisiert. Gerade große Publisher und Studios setzen mittlerweile auf einheitliche Engines, die sie häufig lizenzieren und an ihre speziellen Bedürfnisse anpassen. Dabei hat sich die Enginetechnologie in vielerlei Belangen von dem ursprünglichen Genre der First-Person-Shooter emanzipiert. Die CryEngine 3, ein Abkömmling des First-Person-Shooters FarCry, kommt beispielsweise sowohl in Geschicklichkeitsspielen (Fibble Flick’n’Roll), in online Rollenspielen im Weltraum-Setting (Star Citizen, Homefront: Revolution), in Strategiespielen (Civilization Online) und für die Produktion von Film- und VR-Medien in Echtzeit (FilmEngine) zum Einsatz. Der Anzahl möglicher Genres von Videospielen ist vielfältig. Weitverbreitete Genres wie Sportspiele, First-Person-Shooter, Real-Time Strategy Games oder andere wie EcoGames, News Games, Art Experiences, Rogue-likes, Tanz- und Musikspiele – die meisten Genres haben spezifische Ansprüche an die zugrundeliegende Engineering Technologie und an die Gestaltungsmöglichkeiten der Content Generation Toolchain (Gregory, 2014). Dass heißt im Umkehrschluss, dass neue Games Engineering Technologien das Potenzial haben, auch neue Genres hervorzubringen.

---

[4] Vor der aktiven Veröffentlichung von Game Engines und ihrer Toolchains hat die Modularisierung bereits eine ambitionierte Gruppe von inoffiziellen Content Creators hervorgebracht - den sogenannten *Moddern* (von engl. Mod*ification*, Veränderung).

Die Vielfalt von Games Engineering Produkten macht es schwer einen dominierenden Trend vorwegzunehmen. Natürlich prägen die Entwicklungen des Hardwaremarktes den Funktionsumfang breit aufgestellter Engines und die Ansprüche spezialisierter Tools. Der derzeitige Trend zur parallelen Berechnung von Simulationsinhalten auf Grafikkarten sowie deren hardwarenahe Programmierung haben beispielsweise maßgeblich zur Schaffung der neuen Vulkan Grafik-API beigetragen (Khronos Group, 2016). Auch die Realisierung massenmarktfähiger Virtual Reality und Mixed Reality Hardware schlägt sich in Games Engineering Technologien durch entsprechende Softwareerweiterungen nieder. Veränderte Spielgewohnheiten erfordern außerdem verstärkt die Synchronisation von Spielzuständen über Kommunikationsnetzwerke und deren Speicherung in der Cloud. Um die komplexen Backendstrukturen plattformübergreifend und einheitlich zur Verfügung zu stellen, kristallisieren sich dafür entsprechende Software-as-a-Service Produkte heraus wie beispielsweise die Google Cloud Platform oder GamingAnywhere.

KERNKOMPONENTEN VON GAME ENGINES

Game Engines bilden den softwaretechnischen Kern von Spielen, indem sie hochspezialisierte Komponenten für unterschiedliche Aspekte von Spielen vereinen und in einem einheitlichen Rahmen bereitstellen (Thorn, 2010). Das Erfassen von Benutzereingaben (*Input*) ist unabdingbar. Die Darstellung grafischer Inhalte (*Rendering*), die Simulation physikalischer Effekte (*Physics*), und die Berechnung und Wiedergabe realistischer räumlicher Audioeffekte (*Audio*) sind integrale Bestandteile fast aller Game Engines. Von keiner geringeren Bedeutung ist eine Infrastruktur für die effiziente Handhabe von Inhalten (*Asset Management*). Diese Inhalte können vielfältiger Natur sein und umfassen unter anderem grafische Daten wie Bilder, dreidimensionale Geometrien oder Animationsverläufe, Informationen über Einstellungen virtueller Kameras und Lichtquellen, Sounds und Musikstücke, Beschreibungen physikalischer Eigenschaften virtueller Objekte oder Programmcode (*Scripts*), der Objekte in einer Szene miteinander interagieren lässt. Die Integration von Netzwerkdiensten (*Networking*) für Mehrspielerkontexte ist ebenso Teil des Kernbereichs, und in neueren Engines auch die Unterstützung von Künstlicher Intelligenz (*Artificial Intelligence*), beispielsweise zur Pfadfindung von Nicht-Spieler-Charakteren.

Input

Das Input-Subsystem muss die Eingaben des Benutzers in Echtzeit auswerten und verschiedene Geräte über eine einheitliche Schnittstelle erreichbar machen (Thorn, 2010). Neben speziellen Eingabesystemen, wie Nintendos *WiiMotes* oder Microsofts *Kinect*, waren weit verbreitete Eingabemechanismen bis vor kurzem weitestgehend auf analoge und digitale Auswahl- oder Reglerwerte (Maus, Tastatur, Joysticks) beschränkt. Seit dem Aufkommen von Smartphones und Tablets hat die Multi-Touch-Eingabe gerade den Sektor Mobile Games nachhaltig dominiert. Durch den anhaltenden Trend von VR und AR Anwendungen gewinnt exaktes *Tracking* von Händen, bald auch von Körperhaltung und Fingern und die damit einhergehende Gestenerkennung zunehmend an Bedeutung.

Um Benutzereingaben zu verarbeiten, ist es notwendig, diese in den Programmfluss einzuschleusen. Je nach Modalität geht dem ein mehr oder weniger aufwändiger Vorverarbeitungsprozess voraus: Rauschen muss gefiltert, partielle Daten mit plausiblen Werten ergänzt, und, gegebenenfalls,

die resultierenden Daten interpretiert werden. Die Vorverarbeitung fällt größtenteils in den Aufgabenbereich hardwarespezifischer Gerätetreiber, die anschließende Bereitstellung in den Aufgabenbereich der Betriebssysteme. Auf Betriebssystemebene werden Nachrichten und Warteschlangen (*Message Queues*) verwendet, um in einem ersten Schritt von der Hardware (*Interrupts*) zu abstrahieren. Applikationen hängen sich in dieses System ein und reagieren auf das Eintreffen neuer Nachrichten (bspw. mit *Callbacks*) oder fragen selbst nach neuen Nachrichten (*Polling*). Letzteres erfordert eine stringenter getaktete Einflussnahme des Anwenders auf das System, limitiert also unter Umständen dessen Reaktionszeit, bietet aber gleichzeitig eine höhere Kontrolle über den tatsächlichen Programmablauf, da zufällige Unterbrechungen laufender Prozesse vermieden werden.

Graphics

Das Rendering-Subsystem bzw. die *Graphics Engine* ist für die Umsetzung von Szeneninhalten und -zuständen in gerenderte Bilder verantwortlich (Thorn, 2010). Graphics Engines stellen oft eigene Formate für Texturen, Animationen, Videos und Modelle bereit, die auf Effizienz getrimmt sind, sowie eigene Importpfade, sodass ausgereifte Modellierungs-Tools wie *Autodesk 3ds Max*, *Maya* oder *Blender* verwendet werden können. Die Programmierung von eigenen visuellen Effekten durch *Shader* kann ebenfalls von der Game Engine unterstützt werden. Shader sind kleine Programme, die auf der Grafikkarte ausgeführt werden. Unreal Engine 4 bietet beispielsweise eine visuelle Programmierung auf Basis der *High-Level Shader Language* (*HLSL*) an.

Effekte wie globale Beleuchtung (*Global Illumination)* oder *Final Gather (FG)*, welche die Auswirkungen von indirekter Beleuchtung zwischen verschiedenen Objekten in einer Szene umsetzen, sind technisch anspruchsvoll und gleichzeitig von großer Wichtigkeit für eine realistische visuelle Darstellung. Präzise Schlagschatten oder Schatteneffekte wie Umgebungsverdeckung (*Ambient Occlusion*) sind ebenfalls komplex und aufwendig. Für manche Effekte, die noch nicht in Echtzeit berechnet werden können, bieten aktuelle Game Engines häufig die Möglichkeit der statischen Vorberechnung (*baking*) an. Spezielle Effekte wie Funkenflug, Feuer oder Rauch werden über sogenannte Partikelsysteme realisiert, die in beschränktem Umfang physikalische Modelle realisieren.

Physics

Das Physik-Subsystem bzw. die *Physics Engine* sollte eine *plausible* Simulation des Verhaltens von virtuellen, physischen Objekten in der virtuellen Welt erlauben (Thorn, 2010), wobei man dabei vorrangig die Simulation klassischer Mechanik von Festkörpern im Blick hat (Millington, 2007). Damit lassen sich grundlegende Effekte wie Gravitation, Kollisionen und Reibung verwenden, um interaktive und plausible Welten zu erzeugen: Kisten können geworfen werden, Kugeln folgen ballistischen Trajektorien. Werden zwischen den Körpern mathematische Einschränkungen definiert (*constraints*), kann man physikalische Abhängigkeiten modellieren: So bewegt sich bspw. ein (ganzes) Mobile im Wind, obwohl nur eines seiner Teile von ihm erfasst wurde. Diese Abhängigkeiten können selbst als Spielemechanismus verwendet werden, wie man es plakativ bei diversen „Bridge Construction“ Spielen erleben kann. Umgekehrt kann durch constraints der Zusammenhalt von Objekten definiert werden, die im Verlauf eines Spiels in ihre Einzelteile zerlegt werden können (*NVidia Blast*). Weiche, verformbare Körper wie Tücher für Flaggen und Kleidung, Haare oder Seile sind mittlerweile ebenfalls allgegenwärtig (*NVidia FleX PhysX*, *NVidia HairWorks*). Auch Simulationen von Flüssigkeiten und Gasen (insbesondere Rauch) sind bereits (abstrahiert) in Echtzeit möglich (*NVidia FleX PhysX*). Im Gegensatz zu wissenschaftlichen Simulationen wird häufig Präzision der Effizienz untergeordnet und nur „visuelle Plausibilität” gefordert. Durch die weite Verbreitung *General Purpose Computation on Graphics Processors* (*GPGPU*) fähiger Hardware werden mittlerweile auch Physics Engines Hardware-effizient auf Grafikkarten berechnet, was viele physikalische Effekte in Echtzeit erst möglich macht.

Audio

Das Audio-Subsystem ist von großer Bedeutung für die Erzeugung immersiver Szenen. Techniken wie *Dolby Digital Live* oder *DTS Connect* erlauben es, Mehrkanalton in Echtzeit zu generieren. Eine andere, aktuelle Fragestellung, die insbesondere durch VR Anwendungen an Aufmerksamkeit auf sich gezogen hat, ist die korrekte Berechnung der Ausbreitung von Schallwellen in Abhängigkeit der (simulierten) Kopfposition und der akustischen Eigenschaften des virtuellen Raumes. Für die korrekte Simulation eines solchen Raumklangs gibt es verschiedene Ansätze. Präzise nutzer-

zentrierte Verfahren wie etwa die Berechnungen einer *Head-Related Transfer Function* stellt aktuelle Hardware noch vor große Herausforderungen.

Ressourcenmanagement

Das Ressourcenmanagement organsiert und orchestriert den Zugriff auf prinzipiell beschränkte Hardware- wie Software-Ressourcen des Spielsystems. Dies betrifft zentral die Ressourcen Rechenzeit, Speicher und Peripheriezugriff. Die zu verwaltenden digitalen Medieninhalte heutiger Spiele können sehr umfangreich werden. Da einerseits (Arbeits-)Speicher eine begrenzte Ressource und andererseits das Nachladen von Daten von großvolumigen Datenträgern mit hoher Latenz verbunden ist, müssen Maßnahmen ergriffen werden, um Ressourcen eindeutig zu identifizieren, diese schnell bereitzustellen, gegebenenfalls Speicher wieder freizugeben und das mehrfache Laden identischer Ressourcen zu unterbinden. Hierbei können Asset-Datenbanken, Engine-spezifische Formate und die Vorverarbeitung von Medieninhalten zur Anwendung kommen (Thorn, 2010).

Die Verteilung von notwendigen Algorithmen auf die verfügbare Rechenzeit hat wiederum einen signifikanten Einfluss auf die zeitlichen Anforderungen des jeweiligen Spiels. Benutzereingaben haben einen Einfluss auf die Simulation des Spielzustands sowohl in Bezug auf die Konsistenz als auch in Bezug auf etwaige Berechnungszeiten (man denke etwa an sich verändernde Kameraperspektiven). Dieses führt zu teils schwer vorherzusehenden Lastschwankungen, insbesondere bei umfangreichen Daten und deren Verarbeitung. Daneben verfügen heutige Spielsysteme in der Regel über Mehrkernarchitekturen und dedizierte Hardware für bestimmte Berechnungen verfügen (siehe heutige GPUs). Die Vielzahl verschiedener Algorithmen innerhalb der Spielsimulation optimal auf die verfügbaren verteilten Rechenzeiten zu verteilen, etwaige Lastschwankungen abzufangen und gleichzeitig zeitliche Randbedingungen einzuhalten bedingt ein weitreichendes zeitliches Ressourcenmanagement, ein Scheduling.

Scripting

Eine *Scripting Engine* ermöglicht die schnelle Umsetzung komplexer, interaktiver Spielelogik. Das schließt nicht nur die Mechanismen ein, die den Videospieler herausfordern, bspw. wie man einen virtuellen Fußballspieler führen muss, damit er nicht den Ball verliert, sondern auch mögliche Verhalten von NPCs oder Scoring Mechanismen. Scripting ermöglicht zumeist

auch, technische Aspekte des Spielprozesses zu steuern, also zum Beispiel Netzwerkverbindungen aufzubauen oder bestimmte Ressourcen zu laden.

Die Möglichkeit, sich auf die Definition der Spielelogik zu fokussieren, kommt insbesondere Game Designern und Art Designern entgegen, da der Kreativprozess nicht mit Details der technischen Implementierung überladen wird. Scripting unterscheidet sich hier von der restlichen Programmierung mit oder in der Engine insbesondere dadurch, dass es auf den Ressourcen der Engine aufbaut und über eine klar definierte Schnittstelle damit interagieren kann, selbst jedoch nicht Teil des Engine-Codes ist. Insbesondere ist keine zeitaufwendige Kompilierung der gesamten Plattform vor der Ausführung eines neuen oder geänderten Skripts nötig.

Animation

Interaktive Erlebnisse erfordern eine dynamische Gestaltung der virtuellen Welt. Die bloße Bewegung eines geometrischen Objekts haucht ihm Leben ein, es wird „animiert". Insbesondere die natürlich anmutende Fortbewegung von Lebewesen, allen voran Humanoiden, ihre lebendige Körpersprache und ihre physischen Interaktionen mit der virtuellen Welt sind für ein glaubhaftes Spieleerlebnis wichtig.

Man erreicht die Animation eines komplexen Körpers mit einer Vielzahl von Gliedmaßen dadurch, dass man sie an Knochen (*bones*) bindet, die wiederum mittels Gelenken (*joints*), realisiert durch Einschränkungen der constraints, verbunden werden (*rigging*). Bewegt man nun einzelne Gliedmaßen des daraus resultierenden *artikulierten Körpers,* bewegen sich die davon abhängigen Gliedmaßen mit. Damit sich die geometrischen Teilabschnitte des Körpers nicht ungünstig überlappen, wenn sie bewegt werden, wird ihre Beweglichkeit im Vorfeld festgelegt (*skinning*). Auch ist es üblicherweise notwendig, sicherzustellen, dass die Geometrie mit gleichmäßiger Qualität und Auflösung definiert wird, um Artefakte bei den Bewegungen zu vermeiden. Egal ob die komplexen Animationen manuell definiert (*keyframing*), durch die Verwendung von Motion-Tracking Systemen aufgenommen oder gar prozedural generiert wurden, es muss wohldefiniert sein, wann welche Geometrie wie im Raum platziert sein muss. Diese Information kann in Modellierungsprogrammen oder dedizierten Editoren in Game Engines in sogenannten *Animationskurven* festgehalten werden. Um das Abspielen von Animationen abhängig vom Zustand des virtuellen Objekts zu machen, bieten ausgereifte Game Engines an, Zustandsautomaten

(*State Machines*) zu modellieren. Von einem Anfangszustand ausgehend legen sie fest, welches Ereignis zu welchem Nachfolgezustand eines Objekts im Spiel führt und gegebenenfalls auch, welche Animation damit verknüpft sein soll.

Artificial Intelligence

Damit ein Spiel dauerhaft interessant bleibt, muss der Spieler Spaß dabei empfinden (Koster, 2013). Dabei muss man darauf achten, dass der Spieler nicht unterfordert wird und sich langweilt, noch dass er überfordert und frustriert wird (Csikszentmihalyi, 1990). Schafft man es, dass man die *Kompetenz* des Spielers über das Spielgeschehen hinweg adäquat fordert, muss man nur noch sicherstellen, dass er prinzipiell einen Bezug (*relatedness*) zu dem Spiel aufbauen kann - ob über seine Mitspieler oder eine interessante Geschichte - und dass man ihm die Freiheit gewährt, das Spielgeschehen zu erkunden und genießen (*autonomy*). Um diese Aspekte adäquat zu adressieren, ist es oftmals notwendig, die virtuelle Welt intelligent auf den Spieler reagieren zu lassen, um sicherzustellen, dass NPCs sich *plausibel* verhalten und dass der Schwierigkeitsgrad maßvoll zunehmen kann.

Die dafür notwendige Funktionalität kann nur sehr beschränkt mit einfachen Skripten erzielt werden: Man stelle sich eine große Anzahl von NPCs vor, die sich über ein Terrain bewegen - die Pfade, die sie wählen, müssen plausibel sein. Sofern sie alle den gleichen wählen, müssen sie ihr Fortkommen zusätzlich (lokal) koordinieren. Oder man stelle sich vor, man wollte neue Levels generieren, um dem Spieler neue Inhalte präsentieren zu können und somit die Attraktivität zu steigern, ein Videospiel mehrfach zu spielen oder um die spezielle Spielsituation auf die Fähigkeiten und Erfahrungen des Spielers abzustimmen (*player modelling*). Man müsste aufgrund einiger weniger Variablen neue Level automatisch generieren. Die gleichen generativen Techniken (*procedural content* generation, PCG) können auch zum Einsatz kommen, um die virtuelle Welt – eventuell bereits während der Designphase – vielfältiger zu gestalten, indem man beispielsweise einzelne Geometrien oder Texturen mit hoher Varianz generiert.

Traditionell bieten *Artificial Intelligence (AI) Engines* Möglichkeiten, plausible Verhalten von NPCs zu definieren, beispielsweise über Zustandsautomaten, ähnlich derer für die Definition von Animationsübergängen (Millington und Funge, 2016). Der Kern vieler AI Engines besteht darin, plausible Pfade für Avatare und NPCs in komplexen Terrains zu generie-

ren. Plausibel bedeutet in diesem Kontext üblicherweise, dass man nicht gewillt sein sollte, große Umwege in Kauf zu nehmen, vor allem nicht, wenn der kürzeste Weg offensichtlich scheint. Bei der Bewertung eines Pfades muss man natürlich auch weitere potentielle Erschwernisse bedenken, also beispielsweise Höhenmeter oder schweres Gelände, sowie mögliche Vorteile, die sich durch Umwege ergeben, beispielsweise das Aufsammeln von Ressourcen.

Plausible Pfade werden prinzipiell dadurch gefunden, indem man von einem Anfangspunkt aus alle nachfolgenden Wegpunkte in Erwägung zieht, bis man zum Ziel gelangt und sich dann für den sinnvollsten Pfad entscheidet. Das Netz der Wegpunkte begreift man wiederum als Graphen (siehe den Abschnitt zu Szenengraphen), wobei ein valider Pfad eine Sequenz von Knoten von einem Anfangs- zu einem Endknoten darstellt, ohne dass man einen Knoten zweimal besucht. Je besser die Suche nach dem besten Pfad gerichtet ist, also je weniger Wegpunkte man erkunden muss, desto schneller kann man den gewünschten Pfad berechnen, desto längere Pfade oder desto mehr parallele Pfade kann eine AI Engine gleichzeitig finden.

AI Engines können auch Möglichkeiten für PCG zur Verfügung stellen (Yannakakis und Togelius, 2017), es NPCs ermöglichen, allgemein sinnvolle Entscheidungen zu treffen, bspw. auf Grundlage sogenannter *Entscheidungsbäume* oder *logischer Schlüsse*. Um sich einerseits konsequent auf die Fähigkeiten des Spielers einstellen zu können und andererseits, um große Variabilität des Spiels zu erreichen, spielen Verfahren des Maschinellen Lernens eine zunehmend große Rolle: das Spiel beobachtet dabei die Aktionen des Spielers, stellt fest, welche Taktik oder Strategie er verfolgt, welche Levels er besonders gerne spielt, oder wo er Probleme hat und der Spielfluss gefährdet wird. Von derartigen Beobachten kann eine AI Engine dann ableiten, welche zukünftigen Herausforderungen, welche Level Designs, oder welche Hilfestellungen gebraucht würden, um den Spielern ein optimales Erlebnis zu ermöglichen.

BASISTECHNOLOGIEN DES GAMES ENGINEERING

Das Schaffen virtueller Welten und damit auch die Entwicklung von RIS und Game Engines wurde maßgeblich durch Forschung und Entwicklung im Bereich des Datenmanagements und der Computergraphik vorbereitet und vorangetrieben. In diesem Abschnitt gehen wir konkret auf Entwurfsmuster und Datenstrukturen ein, die verwendet werden, um die vielseitigen und vielfachen Interaktionen in RIS effizient zu organisieren.

Codestruktur mit OOP und ECS

Die *objektorientierte Programmierung* (OOP) ist eines der prominenten Programmierpradigmen der letzten Jahre. Alternative Ansätze wie das funktionelle Paradigma geraten momentan mehr und mehr in den Fokus, dennoch ist OOP momentan noch bei vielen Entwicklungen im Games Engineering wie in der generellen Softwareentwicklung das Paradigma der Wahl für große und komplexe Softwaresysteme. OOP gibt dem Entwickler

Möglichkeiten an die Hand, seinen Code zu organisieren und wieder zu verwenden. Objekte sind dabei als Einheiten von Daten und Operationen zu sehen, die durch klar definierte Schnittstellen mit anderen Objekten interagieren. Definiert man eine Klasse von Objekten, bspw. Automobile, können Unterklassen, bspw. Lastwagen oder Rennautos, die Attribute und Methoden der Oberklasse erben, also wiederverwerten oder gegebenenfalls anpassen und erweitern. Das Prinzip der objektorientierten Programmierung stellt die Kapselung von Daten, die ein Objekt definieren, und Methoden, die diese Daten manipulieren, in den Mittelpunkt des Designs von Software (Gamma, 1995). Die sich dadurch ergebende funktionale Trennung zwischen verschiedenen Objekten erhöht unter anderem die Wartbarkeit in großen Projekten.

Folgt man dem klassischen, objektorientierten Ansatz, könnte man beispielsweise folgende Hierarchie von Objektklassen entwerfen, um eine virtuelle Welt mit Datenobjekten zu bevölkern. Zunächst einmal könnte man eine Basisklasse *GameObject* definieren, die einem beliebigen Objekt nur einen Namen zuweist. Weiterhin könnte man eine Unterklasse *SpatialObject* einführen für jene Objekte, die einen konkreten Platz in der virtuellen Welt einnehmen, also Attribute für ihre Position und Orientierung benötigen. Eine weitere Unterklasse *GeometricObject* könnte geometrische Daten mit dem Objekt assoziieren, um es nicht nur benennen und platzieren, sondern auch visuell darstellen zu können. Eine weitere Unterklasse *PhysicsObject* könnte das Objekt um eine physikalische Repräsentation erweitern, damit Kräfte auf das Objekt einwirken und bspw. seine Position ändern können. Physikalische Eigenschaften können auch andere Auswirkungen haben. Beispielsweise kann eine Physics Engine auch Kollisionen zwischen Objekten feststellen. Wollte man nun ein Objekt kreieren, das zwar derartige physikalische Kollisionen feststellen kann, aber keine geometrische Repräsentation besitzt, wäre diese Hierarchie nicht mehr zielführend: obwohl das Objekt auch von der Oberklasse GeometricObject erbt, besäße es keine Geometrie. Eine sekundäre Hierarchie müsste eingeführt werden, um konsistent zu bleiben (GameObject → SpatialObject → *InvisiblePhysicsObject*). Es würden nun offensichtlich bestimmte Attribute mehrfach in alternative Hierarchien eingeführt werden. Die Wartbarkeit sinkt - da man für Änderungen der physikalischen Attribute und Methoden nun mehrere Objektklassen anpassen müsse. Eine Mehrfachvererbung, also das gleichzeitige Erben von mehreren Elternklassen, könnte im genannten Beispiel Abhil-

fe schaffen - indem eine *VisiblePhysics*-Klasse sowohl von GeometricObject als auch von InvisiblePhysicsObject erbt. Allerdings verursacht Mehrfachvererbung weitere Probleme. Insbesondere könnten gleichlautende Attribute und Methoden zu großen Problemen durch den Verlust der Eindeutigkeit von Zuständigkeiten (*Diamond Pattern*) führen. Da der objektorientierte Ansatz generell schnell dazu führen kann, dass vielfach und vielschichtig Abhängigkeiten und Konflikte zwischen Objektklassen eingeführt werden, ist es oftmals auch schwierig, große Codebasen weiter zu entwickeln und zu pflegen (Thorn, 2010). Daher wird eine derartige Modellierung für das gegebene Problem heute eher als *Anti-Pattern* gesehen. Die Nachteile wiegen die Vorteile auf.

Eine Lösung besteht darin, den Blickwinkel zu ändern. Statt einer komplexen Vererbungshierarchie können die Elemente in der Spielwelt auch als eine Kombination von Eigenschaften gesehen werden, die jeweils für verschiedene Aspekte (Rendering, Physik, Audio,…) relevante Zustandsbeschreibungen liefern. Im entsprechenden *Entity-Component-System* (ECS) Entwurfsmuster werden alle Objekte der virtuellen Welt (*entities*) als Aggregation verschiedener Datenobjekte (*components*) dargestellt. Die einzelnen Komponenten werden mit unabhängigen Subsystemen der Game Engine registriert, von ihnen ausgelesen und manipuliert.

Auf den ersten Blick mag dies den Paradigmen der OOP widersprechen. Tatsächlich steht es jedoch als Entwurfsmuster nicht in Widerspruch zu einer konkreten objektorientierten Implementierung der Game Engine. Das Prinzip der Aggregation wird über das der Vererbung gestellt. Zugriffe mehrerer Engines auf ein und denselben Datenblock, bspw. die Positionierung und Orientierung eines Objekts, können durch eine vorgegebene Ausführungsordnung der Engines koordiniert und eine klare Aufgabentrennung zwischen den Engines erzielt werden. Das Ideal der vollständigen Trennung und Entkoppelung ist allerdings nicht immer umsetzbar. Als eine Variante des *Data-Driven Design* hat ECS nachweislich seit Ende der 1990er Jahre in Game Engines Einzug gehalten (*Thief* 1999, *Dungeon Siege* 2001). Die Game Engine *Unity3D* sowie die RIS Plattform *Simulator X* sind Beispiele dafür, wie man ECS in einer Game Engine und in RIS Plattformen effektiv einsetzen kann.

Modularisierung und Subsysteme

Monolithische Engines ziehen eine enge Integration der verschiedenen Berechnungsprozesse vor, um unnötige Zwischenschritte beim Datenzugriff und damit einhergehend Latenzen zu vermeiden. Eine klare Strukturierung der Game Engine in voneinander unabhängige Subsysteme bringt jedoch Flexibilität, die für einen diversen Videospielmarkt vorteilhaft ist und erleichtert die Pflege der zugrundeliegenden Codebasen (Nystrom, 2014). Die Modularisierung hat weiter den Vorteil, dass Funktionalitäten verschiedener Engines frei kombiniert und über grundlegende Bestandteile hinaus für Game Designer und Entwickler angeboten werden können. Aufgrund des Zielkonflikts zwischen Modularität und Performanz finden sich jedoch insbesondere in genrespezifischen Engines oftmals pragmatische Einschränkungen.

Die einzelnen Subsysteme müssen zur Laufzeit zusammengeführt werden. Ein *Managersystem* übernimmt ihre Koordination und stellt die Datenstrukturen bereit, mit denen die Entitäten im Spiel verwaltet, adressiert und traversiert werden können. Zusätzlich ist auch die Entkoppelung von Subsystemen, die mit verschiedenen Geschwindigkeiten arbeiten, von großer Bedeutung. So laufen die drei Hauptkomponenten in vielen interaktiven Simulation und Spielen mit unterschiedlichen Update-Raten: Eine flüssige grafische Darstellung benötigt mindestens 60 Hz (für hochqualitative VR Anwendungen sind 90 Hz derzeit Standard, und 120 Hz bereits in Diskussion), eine konsistente Physics Engine etwa 120 Hz, während KI Entscheidungen oft nur mit 1 Hz getroffen werden müssen. Benutzereingaben kommen wiederum in unregelmäßiger Frequenz im System an.

Echte Nebenläufigkeit von Berechnungen wird durch die steigende Verfügbarkeit von Mehrkernprozessoren stetig attraktiver. Das ermöglicht z.B. einzelne, zeitintensive Tasks, wie die Berechnung eines kürzesten Pfades zwischen zwei Wegpunkten, in einen kleinen, parallel ausgeführten Prozess auszugliedern, um einige wenige Frames später ein optimales Ergebnis zu erhalten. Neben der parallelen Ausführung von Subengines können so auch Teilaufgaben von Physik, Grafik oder Logik gleichzeitig berechnet werden. Zu Ende gedacht würden alle Teilaufgaben von sämtlichen Subengines auf alle zur Verfügung stehenden Prozessoren derart verteilt, dass sie sie diese maximal auslasten, nach Priorität ausgeführt und bestmöglich synchronisiert würden. Eine Vielzahl nicht-trivialer Designentscheidungen und offener Forschungsfragen stehen dieser Ambition derzeit

noch im Weg. Gleichwohl hat die Forschung bereits einen langen Weg in Richtung RIS und Game Engines zurückgelegt.

Von Bildgeneratoren und Graphikbibliotheken

Die industriellen Ursprünge moderner interaktiver Bilderzeugung lassen sich auf die von Ivan Sutherland und David C. Evans 1968 gegründete Firma *Evans & Sutherland* zurückführen, die als eine der ersten Spezialisten für militärische und industrielle Trainingssimulatoren sowie hochperformante Grafikchips für echtzeitfähige Darstellungen, sogenannte *Image Generators,* auftrat. Ihr Image Generator *CT3* führte 1976 erstmalig die Trennung von visuellen Daten (*Visual Database*) und der Darstellungssoftware ein, wodurch sich die Verarbeitung der Daten durch drei Stufen definieren ließ: *Visual Database*, *Image Generator* und *Display*. Der CT3 konnte 900 Polygone (Vielecke) bei 25Hz darstellen, verfügte über Kantenglättung und Gouraud-Shading, eine Methode zur automatisierten Schattierung geometrischer Flächen (Christianson, 1989). 1979 gründete James Clark die Firma *Silicon Graphics Inc.* (später *SGI*). SGI hat für ihre dedizierten *IRIS Graphics Workstations* die Applikationsbibliothek *IRIS GL* (*Graphics Library*) entwickelt, mit der sich graphische Daten z. B. Geometrien präzise definieren und auf spezialisierter Hardware verarbeiten lassen.

Die IRIS GL Schnittstelle ermöglicht das Zeichnen einzelner Polygone zur Laufzeit einer Applikation durch sofort ausgeführte Darstellungsbefehle (sogenannter *immediate mode*) sowie die persistente Definition von veränderlichen Szenen, die dann kontinuierlich gezeichnete werden (sogenannter *retained mode*). SGI hat für die Umsetzung der Befehle *Rasterizer* entwickelt, spezialisierte Hardware, die geometrische Objekte wie Linien und Polygone effizient als Farbpunkte auf den Bildschirm projizieren. IRIS GL wurde zur hardwareunabhängigen Bibliothek *OpenGL* weiterentwickelt und 1992 veröffentlicht. SGI hat den Renderprozess in einzelne Phasen (*Stages*) unterteilt, die eigenständige Verarbeitungsschritte der Grafikdaten trennen. Diese schrittweise Ausführung der Daten ermöglicht weiterführende Optimierungen des Renderingprozesses und bildet die Grundlage moderner Renderpipelines.

Anfang der 90er Jahre etablierte sich die Firma 3Dfx auf dem Markt, die mithilfe der Expertise ehemaliger SGI Mitarbeiter Grafikbeschleuniger für den Massenmarkt entwickelte. 3Dfx fixierte die modularen, komplexen Stages als fest vorgegebene *Renderpipeline*. 3Dfx lieferte seine eigene Gra-

fikbibliothek *Glide*, die effektiv eine für Spiele optimierte Untermenge der Funktionalität von IRIS GL / OpenGL implementiert. Nach anfänglichen Erfolgen auf dem Videospielmarkt und nachdem OpenGL den Mainstream erreichte, wurden die Einschränkungen von Glide deutlich. OpenGL bot umfangreichere Möglichkeiten und wurde von Konkurrenzprodukten wie der NVidia Riva TNT 2 Grafikkarte vollständiger unterstützt, weswegen es sich letztlich behaupten konnte (Windeck, 1999).

1995, zeitgleich mit Glide, stellte Microsoft *DirectX* vor, eine auf das Windows Betriebssystem[5] beschränkte Schnittstelle zur Implementierung von Multimediainhalten (Theuerjahr, 2007). Sie gewann, auch wegen des hohen Marktanteils von Windows, rasch große Bedeutung für den Spielemarkt. Mit Version 7 von DirectX (1999) wurde die Renderpipeline durch hardwaregestützte Vektor- und Matrizenberechnungen und Beleuchtungsmodelle (*Hardware Transform and Lighting)* erweitert. Diese Erweiterungen mittels fest verdrahteter Berechnungsfunktionen für spezifische Algorithmen der Bilderzeugung bildeten den ersten Schritt in Richtung programmierbare Shader, in welcher die feste Verdrahtung unter Erhalt des Speicher und Datenmodells sukzessive durch eine Programmierbarkeit ersetzt wird. Die Flexibilität der Hard- und Software wurde stetig erhöht, bis mit Version 10 alle Shader-Stages frei programmierbar waren - faktisch ist die Trennung der Hardware Stages bei aktuellen Grafikkarten aufgehoben (*Unified Shader Model*). OpenGL unterstützt programmierbare Shader seit Version 2.0 (Kessenich et al., 2004). Die freie Programmierbarkeit von Shadern hat einen neuen Markt für GPUs eröffnet, der wiederum die Entwicklung aktueller GPUs beeinflusst: General Purpose Computation on Graphics Processors (*GPGPU*). Hierbei wird die massive Parallelität, die sich aus dem Design von 3D Grafikpipelines ergibt, für die Berechnung hochgradig parallelisierbarer Probleme aus allen Bereichen der Wissenschaft und Technik eingesetzt.

Seit den frühen 1990er Jahren sind somit OpenGL und DirectX die am Markt dominierenden Grafikschnittstellen für RIS. Aktuell ist wieder eine Nachfrage hardwarenaher Programmierung zur besseren Ressourcennutzung erkennbar. Auch getrieben von limitierter Hardware aktueller Mobil-

[5] Das Betriebssystem Windows 95 läutete außerdem die Ära der parallelen Ausführung mehrerer Prozesse (Multitasking) ein.

geräte wird diesem Trend mit entsprechenden Bibliotheken wie *OpenGL ES* (*Embedded Systems*) (Khronos Group), *Metal* (Apple) oder *Vulkan* (Khronos Group) Rechnung getragen.

Szenengraphen: Datenstruktur virtueller Welten

Szenengraphen stellen eine wichtige Innovation dar, die die Entwicklung von RIS von Anfang an maßgeblich beeinflusst hat. Einen *Graphen* kann man sich wie ein Diagramm eines sozialen Netzwerks vorstellen, in dem die Nutzer als kleine Kreise (*Knoten*) und ihre Beziehungen als Linien (*Kanten*) dargestellt werden. Graphen können für die Beschreibung und Verarbeitung beliebiger Inhalte verwendet werden. Szenengraphen bezeichnen entsprechend Graphen, die die Bestandteile einer virtuellen Szene strukturieren und nach bestimmten Kriterien organisieren. Komplexere grafische Szenen können mittels dieser Datenstrukturen verwaltet werden. Die systematische Übersetzung solcher dauerhafter vorgehaltenen Daten in Grafik kann dann getrennt von der Applikationslogik in einer Grafikbibliothek umgesetzt werden.

Eine Szene kann unterschiedliche Hierarchien haben, je nach dem unter welchem Aspekt man sie betrachtet. Beispielsweise können Szenengraphen räumliche Zusammengehörigkeit ausdrücken, indem ein übergeordneter Knoten ein geometrisches Volumen repräsentiert, das alle untergeordneten Knoten umfasst. Ein derartiger Szenengraph erlaubt, Geometrien, die zu einem bestimmten Zeitpunkt nicht dargestellt werden müssen, effizient vom Renderingprozess auszuschließen (*Culling*). Zusammengehörige Objekte, wie die Räder oder Türen, die an einem Auto befestigt sind und somit als Einheit bewegt werden, können auch in einem Szenegraphen als *Transformationshierarchie* organisiert werden. Die Vererbung von Position und Orientierung der Elternknoten an ihre Kinder stellt sicher, dass zusammengehörende Gruppen von Objekten gemeinsam transformiert werden. Graphen können auch dazu dienen, Interaktionsprozesse zwischen physikalischen Objekten zu optimieren, bspw. indem man mögliche Kollisionen vorwegnimmt oder ausschließt. Wenn die Kanten eines Graphen Abhängigkeiten zwischen Objekten repräsentieren, können Teilgraphen für deren zielgerichtete parallele Berechnung in Multithreading- oder Multiprozessarchitekturen genutzt werden.

Für den Renderingprozess war es lange Zeit wichtig, die aufwendigen Wechsel zwischen unterschiedlichen Einstellungen des Renderings (soge-

nannte *Render States*) zu minimieren, bspw. unterschiedliche Materialeigenschaften. Objekte mit gleichen States wurden daher in die gleichen Teilbäume einsortiert, um gemeinsam abgearbeitet zu werden, ohne den Render State stets ändern zu müssen.

Insgesamt sind die Vorteile durch die Verwendung von Szenegraphen also vielseitig und von großer Bedeutung für die Organisation und effiziente Berechnung von RIS. In den folgenden Paragraphen erläutern wir ein wenig detaillierter zwei bedeutsame Implementierung von SGI.

Optimierung durch Szenengraphen - IRIS Performer

SGI hat mit IRIS Performer (Rohlf und Helman, 1994) eine Implementierung eines Szenengraphen zusammen mit optimierten Bibliotheken bereitgestellt, die insbesondere die Nutzung von Multiprozessorsystemen erleichterte. Der Szenengraph unterstützte die Gruppierung, Transformation und Auswahl von Knoten und deren Iteration mit fortlaufender Zeit. Außerdem ermöglichte er komplexere Operation, wie zum Beispiel das Überblenden von Geometrien oder das situationsbedingte Laden und Darstellen grafischer Daten mit möglichst geringen Speicher- und Berechnungskosten (*Level of Detail* oder *LOD*). Mit Hilfe von *Switch*-Knoten konnten Teilbäume selektiv eingeblendet werden, und *Sequence*-Knoten ermöglichten es, Animationen auf einfache Art und Weise umzusetzen. Auch die mehrfache Instanziierung von Teilgraphen, und somit deren Wiederverwendung, war möglich. IRIS Performer durchläuft den Szenengraph, der von der Anwendungs-Stage (*APP*) bereitgestellt wird, mit dreierlei operationalen Zielen: (1) *ISECT* zur Detektion von Kollisionen zwischen Objekten, (2) *CULL* zur Selektion sichtbarer Geometrien in Abhängigkeit des Blickwinkels auf die Szene sowie der Bestimmung von LOD und der Sortierung nach dem Render State, und (3) *DRAW* für das Senden der Geometrien an das Grafiksubsystem. Der Nutzer konnte mittels Callbacks zusätzliche Berechnungen während der Traversierungen des Graphen durchführen und dadurch weitere anwendungsabhängige Optimierungen realisieren. IRIS Performer teilt die Berechnungen der vier Stages APP, ISECT, CULL, DRAW auf eine oder mehrere Rendering-Pipelines und eine Intersection-Pipeline auf. APP definiert den Anfang jeder Pipeline und bestimmt deren Ausführung. Außerdem führt APP anwendungsspezifischen Code aus und startet nachfolgende Stages in entsprechenden Pipelines. Spezielle Datenstrukturen und Techniken zum Zwischenspeichern von Daten (*Buffering*)

stellen die Effizienz der erstellten Multiprozessapplikationen sicher. Mit IRIS Performer wurde so eine Vielzahl von Technologien eingeführt, die die Erstellung von Szenengraph-basierten RIS ermöglichen und vereinfachen.

Objektorientierte Szenengraphen - IRIS Inventor

Dem IRIS Performer ging das SGI Framework *IRIS Inventor* voraus. Inventor verfolgte den damals noch sehr neuen objektorientierten Ansatz, um die Erstellung interaktiver Szenen einfacher zu gestalten ohne die erreichbare Komplexität einzuschränken. Dadurch konnte man eine hohe Flexibilität und Erweiterbarkeit gewährleisten. IRIS Inventor wurde später in das *Open Inventor Framework* überführt, das nach wie vor aktiv gepflegt wird.

Das Szenengraphmodell des Inventors umfasst geometrische Objekte (*Shapes*), Eigenschaften (*Properties*) und Gruppierungen (*Groups*), sowie Mischformen. Eigenschaften werden in Feldern (*Fields*) von Knoten im Graphen gespeichert und getrennt von Geometrien definiert. Operationen auf Szenengraphen oder Teilen davon, wie das Rendern oder die Auswahl von Objekten durch Benutzerinteraktion (*Picking*), werden durch *Action*-Objekte umgesetzt, welche den Baum durchlaufen, d.h. alle Knoten beginnend bei der Wurzel besuchen und ihre Operation ausführen. Um Dynamik, bspw. in Form von Animationen, umzusetzen, werden *Sensors*, *Field Connections* und *Engines* verwendet. Sensoren ermöglichen es, zur Laufzeit definierte Callback-Methoden aufzurufen, entweder zeitgesteuert oder aufgrund von Datenänderungen. Field Connections erlauben eine Verknüpfung von Fields verschiedener Nodes, wobei Engines grundlegende Operationen auf beliebigen Eingaben durchführen können, wie beispielsweise die Addition von Vektoren. Die Interaktion mit dem Nutzer des RIS wird dadurch erreicht, dass Benutzereingaben abgefangen werden (*Event Handling)*, als *HandleEventAction* den Graphen durchlaufen und jedem Knoten die Möglichkeit gegeben wird, das Ereignis abzuarbeiten, zu modifizieren, und entweder weiterzuleiten oder nicht. Ein spezieller Knoten für die Selektion mehrerer Objekte (*Selection*) erledigt zusätzlich auch das visuelle Hervorheben selektierter Objekte. Eines der erklärten Ziele des Inventors war es, trotz der Flexibilität und verhältnismäßig leichten Bedienbarkeit hohe Performanz zu erzielen. Dies wurde teilweise dadurch erzielt, dass statische Teilbäume in eine für das Rendering optimierte Form überführt wurden (*state caching*). Die notwendigen Mechanismen zur Sicherstellung

eines konsistenten Zustands waren hierbei transparent in das Toolkit eingebaut.

Szenengraphen leisten bis heute einen wichtigen Beitrag, um die Gestaltung und Berechnung räumlicher Modelle einfach und schnell durchzuführen. Ende der 90er Jahre gab es verschiedene Versuche, einen universellen Szenengraphen zu entwerfen. Effektiv würden dabei die Performer- und Inventor-Ansätze zusammengeführt. Die beiden Projekte, die dieses Ziel verfolgten, Cosmo3D und Project Fahrenheit, wurden jedoch nie fertiggestellt. Relevante Implementierungen von generischen Szenengraphen finden sich unter anderem im *Open Inventor Framework*, in der *Virtual Reality Modeling Language* (*VRML*) bzw. deren Nachfolger *X3D*, *Java3D*, *OpenSceneGraph* oder *OpenSG*.

GAME ENGINES IM WISSENSCHAFTLICHEN BEREICH

Eine Plattform, die die oben genannten Subsysteme bzw. Engines einheitlich integriert und losgelöst von den projektspezifischen Details bleibt, ermöglicht es, mit modernsten Technologien zu arbeiten, ohne für jedes Projekt fehleranfällige und komplexe Integrationsarbeit von Neuem leisten zu müssen. Das performanzgetriebene und zielgerichtete Entwickeln neuer Videospiele hat üblicherweise Vorrang vor der universellen Verwendung einer Game Engine, gerade wenn ein Entwicklerstudio seine eigene Engine pflegt oder eine externe Engine stark an die eigenen Bedürfnisse angepasst hat. Gleichwohl haben gerade Game Engines mit allgemeinerem Anspruch zum Aufblühen der Szene unabhängiger Entwickler (*Independent Developer* oder *Indie Devs*) sowie der Anwendung von RIS in der Wissenschaft geführt.

In einem Rückblick auf die im Department of Computer Science des University College London verwendeten Softwarelösungen für die Erstellung virtueller Umgebungen listet Steed (Steed 2008) unter anderem drei Iterationen der Quake Engine sowie die Unreal Engine 2. Viele aktuelle Veröffentlichungen, unter anderem im Bereich der Verhaltensforschung oder Rehabilitation, aber auch Telepräsenzsysteme, nutzen die Unreal Engine (Bounds 2016, Qiu 2016, Lugrin 2012, Latoschik 2016, Carpin 2007) oder Unity3D als Grundlage (Vasser 2017, Habonneau 2012).

Fachverwandte Vernetzung

Das Forschungsgebiet Games Engineering hat generell zum Ziel, RIS-basierte Softwaresysteme zu entwickeln, die der Unterhaltung von Menschen durch Computerspiele im weitesten Sinne dienen. Als universelle Rechenmaschine soll der Computer auch zur universellen Interaktionsschnittstelle für digitale Systeme und Inhalte und hier speziell zum universellen Unterhaltungsmedium gedeihen. Diese Perspektive erforderte anwendungsfokussierte Anstrengungen, die Ansätze und Methoden einzelner Forschungsbereiche zu integrieren wussten. Gleichzeitig eröffneten Ansätze des Games Engineering gerade durch ihre Anwendungsorientierung neue Forschungspfade in fachverwandten Bereichen. Im Kern stand und steht dabei das Wechselspiel zwischen effizienter Datenverarbeitung einerseits und effizienten sowie intuitiven Nutzerschnittstellen andererseits. Schlussendlich urteilen Anwender nicht nur über den nüchtern quantifizierbaren,

funktionalen Nutzen von RIS, sondern auch über seinen Spaß, die Begeisterung und die Faszination, die sich bei ihnen wecken lassen. Daher werden Game Engines im Allgemeinen auch auf Basis grundlegender Methoden der Mensch-Computer Interaktion (Dix, 2009), wie dem Nutzerzentrierten Design (Donald & Draper, 1986), entwickelt. Man unterscheidet entsprechend zwischen Nutzer (Spieleentwickler) und Endnutzer (Spieler) von Produkten, die dem Games Engineering entspringen. Abhängig vom Spaß und Flow-Erlebnis der Spieler werden Games-Engineering-Inhalte verbessert und angepasst (Csikszentmihalyi, 1990; Chen, 2007). Spieleentwickler wiederum müssen auf der Ebene der Softwareschnittstellen, -architektur und Performanz bedient werden. Werden ihre Ansprüche nicht erfüllt, kommt das betrachtete System, je nach Marktlage, wenig zum Einsatz. Daher entscheidet auch die Softwarequalität und, ebenso wichtig, die Erlernbarkeit sowie die Wiederverwertbarkeit (*Reusability*; Wiebusch & Latoschik, 2015, Latoschik & Fischbach, 2014) der Engines und Subengines über die Akzeptanz von Games Engineering Software. Schnittstellen müssen nachhaltig definiert werden, um eine größtmögliche Modularität und Skalierbarkeit und zudem einen hohen Grad an Benutzbarkeit zu gewährleisten.

Besonders an den Subengines lässt sich die Verwandtschaft zu weiteren Kernthemen der Informatik erkennen. Ansätze der Computergrafik sind schon seit langem fester Bestandteil des Designs von Game Engines (Bishop, Eberly, Whitted, Finch, & Shantz, 1998). Performante Render-Verfahren und effiziente Szenengraphen sind, wie auch aus der kurzen Historie hervorgeht (siehe oben), wesentliche Bestandteile einer Engine Architektur (Eberly, 2006; Gregory, 2009) und somit auch wissenschaftliche Kernthemen des Games Engineering. Besonders in Bezug auf die stetig wachsenden Anforderungen an RIS durch zunehmend immersive Technologien besteht das Bestreben, performante parallele Render-Techniken und Render-Pipelines zu entwickeln. Diese wiederum werden in Subengines integriert, um aktuelle Hardwaresysteme durch hardwarenahe Entwicklung bestmöglich auszunutzen und dadurch dem Spieler ein beeindruckendes Erlebnis zu bieten. Beide Aufgabenstellungen verknüpfen Games Engineering mit klassischen Kerngebieten (Computergrafik, Parallele Programmierung, Hardware/Software Co-Design) der Informatik, die im Weiteren auch an Computer Vision und die digitale Bild- und Signalverarbeitung anknüpfen, die speziell im Bereich der Eingabe und Sensorik, beispielsweise dem Er-

fassen von Körperbewegungen durch Tiefenkameras, für Games Engineering von Bedeutung sind. Diese Verknüpfungspunkte verdeutlichen den generellen Bezug von Games Engineering zu den Disziplinen des Visual Computing.

Die Weiterentwicklung von Grafik- bzw. Visualisierungssystemen durch Forschung und Entwicklung im Games Engineering befruchtet darstellungsbezogene Fachbereiche wie die Architekturvisualisierung, die Datenvisualisierung oder die wissenschaftliche Visualisierung von Modellen. So können auch komplexe Modelle, zum Beispiel durch effektreiche und effiziente Shader, besonders realistisch oder zielgerichtet stilisiert dargestellt werden. Dadurch bedient Games Engineering auch die Fachbereiche der Computeranimation, welche zudem durch Forschung und Entwicklung von Animations-Subsystemen wie z.B. State Machines und Charakter-Systemen, die durch die Eingaben des Nutzers in Kombination mit der Spiellogik virtuellen Charakteren und Objekten Leben einhauchen. Durch diese vielseitigen Entwicklungen trägt Games Engineering auch zu neuen Medienformen, wie dem immersiven Journalismus (De la Peña et al., 2010), sowie der digitalen Film- und Medienproduktion bei.

In Bezug auf neue Medien können durch Games Engineering außerdem die Bereiche e-Learning und Serious Games gestärkt werden. In wissensvermittelnden Simulationen werden vielfach klassische Modelle wie das ARCS Modell (Keller, 1983; Keller, 1987) angewandt, welches als motivationales Instruktionsdesign die Schwerpunkte Aufmerksamkeit, Relevanz, Erfolgszuversicht und Zufriedenheit in den Vordergrund stellt. Durch performante Simulationen lassen sich Aktivität und Reaktivität steigern, was wiederum die Echtzeitfähigkeit der Simulation erhöht. Gutes Engineering kann somit zu positiver Verstärkung dieser Faktoren beitragen. Des Weiteren kann eine Schwerpunktsetzung auf die wissenschaftliche Anwendung von Interaktionstechniken den Lernerfolg verbessern (Oberdörfer & Latoschik, 2016).

Speziell im Zusammenhang mit Trainingssimulatoren spielt auch die Einbindung von AI und NPCs eine entscheidende Rolle. Beispielsweise können angehende Lehrer durch simulierte Unterrichtsstunden Verhaltensweise für eine Realsituation trainieren (Latoschik et al., 2016; Lugrin et al., 2016). Durch immersive Darstellung werden die Nutzer „in die Situation" hineinversetzt. Kerninhalt sind jedoch die realistisch handelnden Schüler, die als NPCs störende Verhaltensweisen überzeugend simulieren müssen.

Games Engineering Technologien unterstützen Serious Games für alle erdenkbaren Anwendungsbereiche, beispielsweise auch in der medizinischen Therapie (z.B. von Phobien) und für medizinisches Training (Bohil, Alicea, & Biocca, 2011; Riva, 2005).

Durch Games Engineering Technologien realisierte interaktive Simulationen können auch dazu dienen, wissenschaftliche Theorien zu kommunizieren, sie zu testen und weiterführende empirische und analytische Forschung zu motivieren (von Mammen, Edenhofer & Hähner, 2015). Dadurch ergibt sich eine allgemeine Nähe zu empirischen Bereichen, auch in der Informatik selbst.

Aussagen über die Entwicklung komplexer Systeme zu machen gilt entsprechend als fundamentale Fragestellung interaktiver Simulationen. Als konkretes Beispiel kann man interaktive Simulationen virtueller biologischer Zellen und Gewebe heranziehen (von Mammen et al., 2012). Die komplexen Interaktionen zwischen den Zellen sind an und für sich auf eine relativ geringe Menge von Interaktionsmöglichkeiten zurückzuführen (Zellteilung, Zelltod, Migration,...). Doch die Formation von Gewebe kann nicht ohne weiteres vorhergesehen werden. Durch effiziente Games Engineering Technologien können entsprechende Simulationen durchlaufen werden, um die beobachteten Prozesse nachvollziehen zu können (Däschinger et al., 2017).

An den Beispielen von interaktiven biologischen Zellsimulationen und physikalisch motivierten Modellen wird klar, dass realistische Simulationen auch Gegebenheiten der Physik wie Aggregatzustände, Masse, Kräfte und Bewegungszustände berücksichtigen müssen. Effiziente parallele Verfahren können heutzutage realistisch Zustände und Verhalten, z.B. von Flüssigkeiten beschreiben, was wiederum auch die Vielfältigkeit des Aufgabenspektrums von Games Engineering widerspiegelt und weiterhin die Verknüpfung zu den Naturwissenschaften und anderen Wissenschaftszweigen verdeutlicht.

Interdisziplinäre Vernetzung

Neben den Verwandtschaften zu informatiknahen Fachgebieten stehen andere Wissenschaftszweige in enger Verbindung und interdisziplinärer Abhängigkeit zu Games Engineering. Auf Systemebene besteht aufgrund der hardwarenahen Entwicklung und hardware-spezifischen Programmierung naturgemäß eine Verbindung zu den *Ingenieurwissenschaften*, wie bei-

spielsweise der *Elektronik*, der *Elektrotechnik* oder der *Mechatronik*. Neben der *Fachinformatik* sind diese Wissenschaftszweige maßgeblich an der Systemintegration beteiligt und wirken an der Konzeption neuer Systeme und Systemkomponenten mit, deren Anspruch vielmals von der Medien- und Spieleindustrie in Form von Requirements, oder durch die Nutzer im Sinne einer großen Nachfrage definiert wird. Weiterhin weist besonders die Konstruktion von Ein- und Ausgabegeräten eine Verbindung mit der *Physik* und dem *Maschinenbau*, der *Medientechnik* und dem *Photoingenieurwesen* auf. Der Austausch von Anforderungen an Gerät und Schnittstelle und komplementär der Integration in die Engine beschreibt diese Verbindung. So muss beispielsweise jede Engine auf neuartige Eingabegeräte, Display Technologien, höhere Auflösungen oder stereoskopische VR Displays angepasst werden. Generell stellt Games Engineering keine festen Regeln an das Medium der Darstellung oder die Form des Eingabe- bzw. Ausgabegerätes, so dass auch Displaytechnologien, die verschiedene menschliche Sinne ansprechen - wie beispielsweise haptisches Feedback, oder Eingabemethoden durch Blickbewegungen oder Brain-Computer Interfaces - untersucht werden. In Verknüpfung mit der *Robotik* können hier Roboter als Interaktionsmedium dienen, wie im Falle des Robocup (Kitano, Asada, Kuniyoshi, Noda, & Osawa, 1997).

An den bereits angesprochenen Beispielen der Fachgebiete e-Learning und Serious Games wird deutlich, dass Games Engineering eine Verbindung zu den *Bildungswissenschaften* herstellt, um neue Methoden digitalen Lernens zu untertützen. In Bezug auf e-Learning können Entwicklungen des Games Engineering dabei helfen, sinnvolle Backend-Konzepte zu entwickeln und beispielsweise für online Quiz-Lernspiele die Unterrichtsinhalte prüfen und vertiefen. Weiterhin können immersive Simulationen für Training und Exposition eingesetzt werden. Beispielsweise können durch die virtuelle Simulation von Fahrsituationen, welche die Auswirkung von Alkoholkonsum auf die Sicht und Sinne simulieren (Gaibler, Faber, Edenhofer, & von Mammen, 2015, von Mammen, Knote, & Edenhofer, 2016), die Gefahren von überhöhtem Alkoholkonsum aufgezeigt werden. Der Spieler erlebt eine mögliche Gefahrenlage, ohne sich in wirkliche Gefahr zu begeben, wodurch Fehlverhalten vorgebeugt werden kann. Weiterhin können durch Makro-Simulationen, z.B. Simulationen von Versorgungsnetzwerken, ingenieurswissenschaftliche Inhalte verdeutlicht werden (von Mammen, Hertwig, Lehner, & Obermayer, 2015). Diese realistisch zu si-

mulieren erfordert nicht nur eine ausdefinierte Spielidee, sondern eine hohe Qualität der Simulation, welche durch Games Engineering Kompetenzen abgedeckt werden kann. Simulationen können den Spieler außerdem in andere Perspektiven versetzen. So können Menschen die Perspektive eines Mosquitos einnehmen, um Orientierung und Lokalisation von Insekten besser zu verstehen (Stifter et al., 2016). Die *Biologie* profitiert hier von Kameramodellen und Render-Techniken, die aus dem Games Engineering entstehen (Lv et al., 2013).

Durch diese visualisierten, modellgesteuerten Simulationen trägt Games Engineering außerdem zu Weiterentwicklungen in den Feldern der *Lebens- und Naturwissenschaften* bei. Informationen über Verhaltensweisen können algorithmisch approximiert und simuliert werden. So kann das Verhalten von Fischen (Schikarski, Meisch, Edenhofer, & von Mammen, 2015) oder Ameisen interaktiv beobachtet (Knote, Edenhofer, & von Mammen, 2016), und Schwärme bzw. Schwarmverhalten modelliert und auf verwandte Anwendungsfelder, wie die Robotik, projiziert werden (von Mammen, Lehner, & Tomforde, 2016; von Mammen et al., 2012).

Bereits am Beispiel von Modellintegrationen und Modellsimulation für Quadrotoren (von Mammen et al., 2016) werden hier Anknüpfungspunkte auch zu der Luft- und Raumfahrttechnik deutlich. Bei Flug-, Strömungs-, und Weltraumsimulation auf wissenschaftlicher Ebene spielt die Präzision und Umsetzung des Modells eine große Rolle, die durch die Entwicklung von entsprechenden Subengines und performanter Rechentechnik getestet werden können. Während hier die Visualisierung oft zweitrangig ist, spielt diese in Disziplinen der Architektur eine deutlich größere Rolle. Virtuelle, dreidimensional geplante Räume können durch Game Engines begehbar gemacht werden, wodurch sich die Erschaffer einen Eindruck von Form und Funktion anhand realgetreuer Abbildungen machen können. Weiterführend können aber auch Ansätze aus algorithmischen und prozeduralen Verfahren dazu beitragen, neue Formen und Strukturen in architektonischen Werken zu verankern (von Mammen und Taron, 2012).

Neben den naturwissenschaftlichen, technischen und technisch-gestalterischen Wissenschaften profitiert Games Engineering auch von den *Sozialwissenschaften* und trägt im Gegenzug besonders in Bereichen der Methodik zu deren Fortkommen bei. Beispielsweise kann die Anwendung von virtuellen Realitäten als Methode den empirischen Forschungsprozess stützen (Fox & Bailenson, 2009). Weiterhin können speziell entwickelte

Spiele oder spezielle Modifizierungen von Spielen (Mods et al., 2016) die *Medien- und Kommunikationspsychologie, Rezeptions- und Wirkungsforschung* sowie die Subdisziplin der *Game Studies* in der Erforschung der Auswirkung von interaktiven medialen Inhalten auf den Menschen bereichern. Die *Kommunikationswissenschaften* und *Kommunikationspsychologie* profitieren von einer Erweiterung der Paradigmen Spannweite. Durch Animation, Multi-Nutzer Umgebungen, Virtueller Realität und Avatar- und Agentenkonzepte können Fragestellungen über klassische Methoden wie *Wizard-of-Oz* Paradigmen hinaus, durch kontrollierte Manipulationen untersucht werden (Roth, 2016; Achenbach et al., 2017; Latoschik et al. 2017, Waltemate et al., 2018). Spezifische Engine Entwicklungen können Verhaltensweisen analysieren und modifizieren, so dass die Engine als aktiver sozialer Moderator in den Kommunikationsprozess eingreift (Roth, Latoschik, Vogeley, & Bente 2015). Hinzu kommt, dass Designs zukünftiger Kommunikationsplattformen und sozialer Medien, die sozial immersive Unterhaltungen durch Avatar-Verkörperung (*Avatar Embodiment*) ermöglichen (Roth, Waldow, Stetter, Bente, Latoschik, & Fuhrmann 2016) wesentlich von Entwicklungen der Informatik und des Games Engineering profitieren. Das sich durch eine virtuelle Verkörperung einstellende Phänomen der Illusion, einen anderen Körper zu besitzen (*Illusion of Virtual Body Ownership*) hilft weiterhin, die Wahrnehmungsprozesse von Menschen zu verstehen und unterstützt dadurch die kognitiven Wissenschaften.

In gleichem Zuge profitieren medizinische Disziplinen zu Diagnose, Rehabilitation, Training und Therapie durch virtuelle Technologien (Georgescu, Kuzmanovic, Roth, Bente, Vogeley, 2014) und Entwicklungen des Games Engineering. Neben Training und Behandlung von Phobien und sozialen Störungen werden durch Simulationen Untersuchungen spezifischer Aktivitäten innerhalb der Neurowissenschaften unterstützt (Bohil, Alicea, & Biocca, 2011). Weiterhin kann Games Engineering im medizinischen Bereich, beispielsweise durch interaktive Simulation und Training von Wurzelbehandlungen (von Mammen, Weber, Opel, & Davison, 2015) oder die Modellierung des menschlichen Körpers (von Mammen, Schellmoser, Jacob, & Hähner, 2015) die Ausbildung und Technik unterstützen.

ZUSAMMENFASSUNG & AUSBLICK

Games Engineering ist ein junges Forschungsgebiet mit einer gleichzeitig verhältnismäßig langen Historie. Diese scheinbare Diskrepanz ergibt sich daher, dass Games Engineering ein *integrativer Forschungsbereich* ist, der auf einem reichen Fundus an Vorarbeiten verschiedener Fachbereiche anknüpft und der im Gegenzug fachverwandte und interdisziplinäre Forschungsbereiche bereichert und von ihnen beeinflusst wird.

Das Ziel des Games Engineering ist der Entwurf, die Entwicklung und Verbesserung von Algorithmen, *Engines*, *Plugins*, *Tools* und Entwurfsmustern zur Realisierung Echtzeit-interaktiver Spiele und Systeme unter Benutzung wissenschaftlicher informatischer Methoden. Bisher verzeichnete das Games Engineering die größten wissenschaftlichen Erfolge bei der Zusammenführung verschiedener Subengines (Graphics, Physics, AI, ...), bei informatischen Teilfragen, bspw. hinsichtlich des Einflusses der Systemperformanz auf den Anwender, und bei interdisziplinären Forschungsfragen.

Durch die Systematisierung des Forschungsgebiets Games Engineering sollen weiterhin wichtige große Ziele verfolgt werden. So soll beispielsweise die interaktive Simulation großangelegter Systemmodelle (*large-scale models*) ermöglicht werden. Die bisher strikt separierten Aspekte des Programmierens bzw. Modellierens und des Simulierens sollen zusammengeführt werden (*Modulation*), sodass die Konsequenzen jeglicher Modellveränderung inkrementell aufgezeigt werden (von Mammen, 2016). Die Komplexität der Inhalte, die ein Mensch zu gestalten vermag, soll stetig zunehmen, am besten veranschaulicht durch Fortschritte in den Bereichen der interaktiven Datenanalyse und natürlicher Mensch-Maschine Interaktionen. Gleichzeitig werden interaktive Technologien die Schnittstellen zwischen technischen Systemen und Menschen weiter intensivieren und dadurch die Systemgrenzen Schritt für Schritt abbauen. VR und AR Anwendungen veranschaulichen diesen Trend. Außerdem soll bei all diesen weitreichenden Zielen auch der Spielspaß Einlass behalten - wieso sollten wir darauf verzichten, unsere Zukunft derart zu gestalten, dass sie auch Spaß macht?

LITERATUR

Abrash, M. (1997). Michael Abrash's Graphics Programming Black Book, with CD: The Complete Works of Graphics Master, Michael Abrash. Coriolis group books.

Achenbach, J., Waltemate, T., Latoschik, M. E., and Botsch, M. (2017). Fast gen- eration of realistic virtual humans. In 23rd ACM Symposium on Virtual Reality Software and Technology (VRST), pages 12:1–12:10.

Arnaud, R., & Jones, M. (1999, November). Innovative software architecture for real-time image generation. In Interservice/Industry Training Systems and Equipment Conference (I/ITSC) Conference.

Bellemare, M., Naddaf, Y., Veness, J., & Bowling, M. (2015, June). The arcade learning environment: An evaluation platform for general agents. In Twenty-Fourth International Joint Conference on Artificial Intelligence.

Bethke, E. (2003). Game development and production. Wordware Publishing, Inc.

Bilas, S. (2002). Presentation: "A Data-Driven Game Object System", Game Developer Conference 2002, San Jose. Letzter Zugriff 14.6.2017, http://scottbilas.com/files/2002/gdc_san_jose/game_objects_slides_with_notes.pdf

Bishop, L., Eberly, D., Whitted, T., Finch, M., & Shantz, M. (1998). Designing a PC game engine. IEEE Computer Graphics and Applications, 18(1), 46–53.

Bohil, C. J., Alicea, B., & Biocca, F. A. (2011). Virtual reality in neuroscience research and therapy. Nature Reviews. Neuroscience, 12(12), 752.

Bounds M., Wilson B., Tavakkoli A., Loffredo D. (2016) An Integrated Cyber-Physical Immersive Virtual Reality Framework with Applications to Telerobotics. In: Bebis G. et al. (eds) Advances in Visual Computing. ISVC 2016. Lecture Notes in Computer Science, vol 10073. Springer, Cham

Bundesverband Interaktive Unterhaltungssoftware e.V. (2017). Marktdaten: Zahlen und Fakten zur Deutschen Computer- und Videospiel-Branche. Online: http://www.biu-online.de/marktdaten/#alle, letzter Zugriff Mai 2017.

Charsky, D. (2010). From edutainment to serious games: A change in the use of game characteristics. Games and culture.

Chen, J. (2007). Flow in games (and everything else). Communications of the ACM, 50(4), 31–34.

Christianson, D. C. (1989). History of visual systems in the Systems

Coppa, E., Demetrescu, C., & Finocchi, I. (2014). Input-sensitive profiling. IEEE Transactions on Software Engineering, 40(12), 1185-1205.

Csikszentmihalyi, M. (1990). Flow. Harper and Row. New York.

Däschinger, M., Knote, A., and von Mammen, S. (in press 2017). An evolutionary approach to behavioural morphometrics. In GECCO '17: Proceedings of the 2017 conference on Genetic and evolutionary computation. ACM New York.

De la Peña, N., Weil, P., Llobera, J., Giannopoulos, E., Pomés, A., Spanlang, B., ... & Slater, M. (2010). Immersive journalism: immersive virtual reality for the first-person experience of news. Presence: Teleoperators and Virtual Environments, 19(4), 291-301.

Dix, A. (2009). Human-computer interaction. Springer.

Donald, A., & Draper, S. W. N. (1986). User Centered System Design: New perspectives on human-computer interaction.

Eberly, D. H. (2006). 3D game engine design: a practical approach to real-time computer graphics. CRC Press.

Eberly, D. H. (2010). Game physics. CRC Press.

Elson, M., & Quandt, T. (2016). Digital games in laboratory experiments: Controlling a complex stimulus through modding. Psychology of Popular Media Culture, 5(1), 52-65. doi: 10.1037/ppm0000033

Engineering Simulator. Technical Report. NASA, Houston,

Fox, J., Arena, D., & Bailenson, J. N. (2009). Virtual reality: A survival guide for the social scientist. Journal of Media Psychology, 21(3), 95-113.

Gaibler, F., Faber, S., Edenhofer, S., & von Mammen, S. (2015, September). Drink & drive: A serious but fun game on alcohol-induced impairments in road traffic. In Games and Virtual Worlds for Serious Applications (VS-Games), 2015 7th International Conference on (pp. 1-5). IEEE.

Gallagher, S., & Park, S. H. (2002). Innovation and competition in standard-based industries: a historical analysis of the US home video game market. IEEE Transactions on Engineering Management, 49(1), 67-82.

Georgescu, A. L., Kuzmanovic, B., Roth, D., Bente, G., & Vogeley, K. (2014). The use of virtual characters to assess and train non-verbal communication in high-functioning autism. Frontiers in human neuroscience, 8, 807.

Gregory, J. (2009). Game engine architecture. CRC Press.

Habonneau N., Richle U., Szilas N., Dumas J.E. (2012) 3D Simulated Interactive Drama for Teenagers Coping with a Traumatic Brain Injury in a Parent. In: Oyarzun D., Peinado F., Young R.M., Elizalde A., Méndez G. (eds) Interactive Storytelling. ICIDS 2012. Lecture Notes in Computer Science, vol 7648. Springer, Berlin, Heidelberg

Herz, J. C. (1997). Joystick nation: How videogames ate our quarters, won our hearts, and rewired our minds. Little, Brown & Co. Inc.

J. Jacobson and M. Lewis, "Game engine virtual reality with CaveUT," in Computer, vol. 38, no. 4, pp. 79-82, April 2005. doi: 10.1109/MC.2005.126

Jean-Luc Lugrin, Fred Charles, Marc Cavazza, Marc Le Renard, Jonathan Freeman, and Jane Lessiter. 2012. CaveUDK: a VR game engine middleware. In Proceedings of the 18th ACM symposium on Virtual reality software and technology (VRST '12). ACM, New York, NY, USA, 137-144. DOI=http://dx.doi.org/10.1145/2407336.2407363

Jones, M. M. (1967). On-line simulation. In Proceedings of the 1967 22Nd National Conference, ACM '67, pages 591–599, New York, NY, USA. ACM.

Keller, J. M. (1983). Motivational design of instruction. Instructional design theories and models: An overview of their current status, 1(1983), 383-434.

Keller, J. M. (1987). Development and use of the ARCS model of instructional design. Journal of instructional development, 10(3), 2-10.

Kessenich, J., Baldwin, D., Rost, R. (2004). The OpenGL Shading Language. Technical Report. 3Dlabs, Inc. Ltd.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., & Osawa, E. (1997, February). Robocup: The robot world cup initiative. In Proceedings of the first international conference on Autonomous agents (pp. 340-347). ACM.

Knote, A., Edenhofer, S., & Von Mammen, S. (2016). Neozoa: An immersive, interactive sandbox for the study of competing ant species. In K-

12 Embodied Learning through Virtual & Augmented Reality (KELVAR), IEEE Virtual Reality Workshop on (pp. 5-10). IEEE.

Koster, R. (2013). Theory of fun for game design. O'Reilly Media, Inc.

Latoschik, M. E., & Fischbach, M. (2014). Engineering variance: Software techniques for scalable, customizable, and reusable multimodal processing. In International Conference on Human-Computer Interaction (pp. 308-319). Springer International Publishing.

Latoschik, M. E., Lugrin, J. L., Habel, M., Roth, D., Seufert, C., & Grafe, S. (2016). Breaking bad behavior: immersive training of class room management. In Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology (pp. 317-318). ACM.

Latoschik, M. E., Roth, D., Gall, D., Achenbach, J., Waltemate, T., and Botsch, M. (2017). The effect of avatar realism in immersive social virtual realities. In 23rd ACM Symposium on Virtual Reality Software and Technology (VRST), pages 39:1–39:10.

Leonard, T. (1999). Postmortem: Looking Glass's Thief: The Dark Project. Game Developer Magazine, 6(7).

Leonard, T. (1999). Website: Postmortem: Thief: The Dark Project. Letzter Zugriff 14.6.2017, http://www.gamasutra.com/view/feature/131762/postmortem_thief_the_dark_project.php

Lewis, M. and Jacobson, J. (2002). Introduction. Commun. ACM, 45(1):27–31.

Lugrin, J. L., Latoschik, M. E., Habel, M., Roth, D., Seufert, C., & Grafe, S. (2016). Breaking Bad Behaviours: A New Tool for Learning Classroom Management using Virtual Reality. Frontiers in ICT, 3, 26.

Lv Z, Tek A, Da Silva F, Empereur-mot C, Chavent M, Baaden M (2013) Game On, Science - How Video Game Technology May Help Biologists Tackle Visualization Challenges. PLoS ONE 8(3): e57990. https://doi.org/10.1371/journal.pone.0057990

Marc Erich Latoschik, Jean-Luc Lugrin, Michael Habel, Daniel Roth, Christian Seufert, and Silke Grafe. 2016. Breaking bad behavior: immersive training of class room management. In Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology (VRST '16). ACM, New York, NY, USA, 317-318. DOI: https://doi.org/10.1145/2993369.2996308

Millington, I., & Funge, J. (2016). Artificial intelligence for games. CRC Press.

Narayanan, S. and Rothrock, L. (2011). Human-in-the-loop Simulations: Methods and Practice. Springer.

Oberdörfer, S., & Latoschik, M. E. (2016, November). Interactive gamified 3D-training of affine transformations. In Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology (pp. 343-344). ACM.

Paul, C. (2003). Digital art (p. 29). London: Thames & Hudson.

Pedersen, R. E. (2003). Game design foundations. Wordware Publishing, Inc..

Pharr, M., Jakob, W., & Humphreys, G. (2016). Physically based rendering: From theory to implementation. Morgan Kaufmann.

Preim, B. and Dachselt, R. (2015). Interaktive Systeme: Band 2: User Interface Engineering, 3D-Interaktion, Natural User Interfaces. Springer-Verlag.

Qiu W., Yuille A. (2016) UnrealCV: Connecting Computer Vision to Unreal Engine. In: Hua G., Jégou H. (eds) Computer Vision – ECCV 2016 Workshops. ECCV 2016. Lecture Notes in Computer Science, vol 9915. Springer, Cham

Riva, G. (2005). Virtual reality in psychotherapy: review. Cyberpsychology & Behavior, 8(3), 220–230.

Roth, D., Latoschik, M. E., Vogeley, K., & Bente, G. (2015). Hybrid Avatar-Agent Technology–A Conceptual Step Towards Mediated "Social" Virtual Reality and its Respective Challenges. I-Com, 14(2), 107–114.

Roth, D., Waldow, K., Stetter, F., Bente, G., Latoschik, M. E., & Fuhrmann, A. (2016). SIAM-C - A Socially Immersive Avatar Mediated Communication Platform. In Proceedings of the International Conference on Artificial Reality and Telexistence / Eurographics Symposium on Virtual Environments.

Roth, D. (2016). The Study of Interpersonal Communication Using Virtual Environments and Digital Animation: Approaches and Methodologies. Presented at the 66th Annual Conference of the International Communication Association, Fukuoka, Japan.

Roth, R., Lugrin, J.-L., von Mammen, S., & Latoschik, M.E. (2017). Controllers & inputs: Masters of puppets. In J. Banks (Ed.), Avatar, assem-

bled: The social and technical anatomy of digital bodies (Ch. 29). Bern, Switzerland: Peter Lang.

S. Carpin, M. Lewis, J. Wang, S. Balakirsky and C. Scrapper, "USARSim: a robot simulator for research and education," Proceedings 2007 IEEE International Conference on Robotics and Automation, Roma, 2007, pp. 1400-1405. doi: 10.1109/ROBOT.2007.363180

Schikarski, J., Meisch, O., Edenhofer, S., & von Mammen, S. (2015). The digital aquarist: An interactive ecology simulator. In Proceedings of the European Conference on Artificial Life 2015 (ECAL) (pp. 389-396).

Schilling, M. A. (2003). Technological leapfrogging: Lessons from the US video game console industry. California management review, 45(3), 6-32.

Stifter, C., Edenhofer, S., & von Mammen, S. (2016). Come Fly with Me-Perceive the World through a Mosquito's Senses. In Games and Virtual Worlds for Serious Applications (VS-Games), 2016 8th International Conference on (pp. 1-4). IEEE.

Theuerjahr, U. (2007). Direct3D Realtime Rendering für Computerspiele: DirectX-Programmierung in C++. Roulio Press.

Vasser M, Kängsepp M, Magomedkerimov M, et al. VREX: an open-source toolbox for creating 3D virtual reality experiments. BMC Psychology. 2017;5:4. doi:10.1186/s40359-017-0173-4.

von Mammen, S., Edenhofer, S., & Hähner, J (2015). CoSMoS in the Interactive Simulation Curriculum. In Proceedings of the 2015 Workshop on Complex System Modelling and Simulation (CoSMoS).

von Mammen, S., Hertwig, F., Lehner, P., & Obermayer, F. (2015). Powersurge: A serious game on power transmission networks. In European Conference on the Applications of Evolutionary Computation (pp. 406-417). Springer International Publishing.

von Mammen, S., Knote, A., & Edenhofer, S. (2016). Cyber sick but still having fun. In Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology (pp. 325-326). ACM.

von Mammen, S., Lehner, P., & Tomforde, S. (2016). Evolving a Facade-Servicing Quadrotor Ensamble. In Proceedings of COGNITIVE 2016 the Eighth International Conference on Advanced Cognitive Technologies and Applications (COGNITIVE) (pp. 16-21).

von Mammen, S., Phillips, D., Davison, T., Jamniczky, H., Hallgrímsson, B., & Jacob, C. (2012). Swarm-based computational development. In Morphogenetic Engineering (pp. 473-499). Springer Berlin Heidelberg.

von Mammen, S., Schellmoser, S., Jacob, C., & Hähner, J. (2015). Modeling and Understanding the Human Body with SwarmScript. The Digital Patient: Advancing Healthcare, Research, and Education, 149-169.

von Mammen, S., Schellmoser, S., Jacob, C., and Hähner, J. (2016). The Digital Patient: Advancing Medical Research, Education, and Practice, chapter 11. Modelling & Understanding the Human Body with Swarmscript, pages 149–170. Wiley Series in Modeling and Simulation. John Wiley & Sons, Hoboken, New Jersey.

von Mammen, S., Taron, J. M. (2012, September). A trans-disciplinary program for biomimetic computing and architectural design. In 6th ASCAAD Conference 2012 CAAD| INNOVATION| PRACTICE (p. 141). Bhzad Sidawi.

von Mammen, S., Weber, M., Opel, H. H., & Davison, T. (2015, April). Interactive multi-physics simulation for endodontic treatment. In Proceedings of the Symposium on Modeling and Simulation in Medicine (pp. 36-40). Society for Computer Simulation International.

Waltemate, T., Gall, D., Roth, D., Botsch, M., and Latoschik, M. E. (2018). The impact of avatar personalization and immersion on virtual body ownership, presence, and emotional response. IEEE Transactions on Visualization and Computer Graphics (TVCG), 24(4).

Webber, J. E., Brewster, K. (2016). 11 video game trends that will change the future of the industry. Accessed Online, Juli 2017: https://www.theguardian.com/technology/2016/jul/21/11-video-game-trends-that-will-change-the-future-of-the-industry

Wiebusch, D., Latoschik, M. E. (2015). Decoupling the entity-component-system pattern using semantic traits for reusable realtime interactive systems. In Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), 2015 IEEE 8th Workshop on (pp. 25-32). IEEE.

Windeck, C. (1999). Heise Website: Freud und Leid: Ergebnisse von nVidia und 3dfx. Letzter Zugriff Juni 2017, https://www.heise.de/newsticker/meldung/Freud-und-Leid-Ergebnisse-von-nVidia-und-3dfx-23745.html

Yannakakis, G. N. and Togelius, J. (2017). Artificial Intelligence and Games. Springer.

SOFTWARETITEL

3ds Max (Autodesk, https://www.autodesk.de/3ds-Max/)
Blender (https://www.blender.org)
CaveLib (Mechdyne, https://www.mechdyne.com)
Civilization Online (Take Two Interactive Software, https://civilizationonline.com/)
Cloud Solutions (Google, https://cloud.google.com/solutions/gaming/)
CryEngine 3 (CryTek, http://www.crytek.com/cryengine/cryengine3/overview)
Crysis 3 (EA/CryTek, http://www2.ea.com/crysis-3)
Doom (id Software, 1995)
Dungeon Siege (Gas Powered Games / Mad Doc Software / Microsoft, 2002)
FarCry (CryTek/Ubisoft, 2004)
Fibble Flick'n'Roll (CryTek, http://www.crytek.com/games/fibble/overview)
FilmEngine (http://www.filmengine.com/)
GamingAnywhere (http://gaminganywhere.org/)
Glide (3Dfx, out of business)
Google Cloud Platform (https://cloud.google.com/)
HairWorks (NVidia, http://www.nvidia.de/object/nvidia-hairworks-de.html)
idTech 1 (id Software, 1993)
idTech 2 (id Software, 1997)
IRIS Inventor (Strauss, 1993)
IRIS Performer (Rohlf und Helman, 1994)
Java3D (https://java3d.dev.java.net)
Maya (Autodesk, https://www.autodesk.de/products/maya/overview)
Metal (Apple, https://developer.apple.com/metal/)
MS-DOS (Microsoft)
NVidia Blast (NVidia, https://developer.nvidia.com/blast)
NVidia FleX Physics (NVidia, https://developer.nvidia.com/flex)
OpenGL (Khronos Group, https://www.opengl.org/)

Open Inventor (http://oss.sgi.com/projects/inventor)
OpenSceneGraph (http://www.openscenegraph.org)
OpenSG (http://www.opensg.org)
PhysX (Nvidia, http://www.geforce.com/hardware/technology/physx)
Quake 2 (id Software, 1997)
Simulator X (Beuth Hochschule für Technik Berlin, http://public.beuth-hochschule.de/~rehfeld/simulatorx.html)
Thief (Looking Glass Studios / Eidos Interactive, 1998)
Universal Windows Platform (Microsoft, https://docs.microsoft.com/de-de/windows/uwp/get-started/universal-application-platform-guide)
Unreal Engine 2 (EPIC, https://docs.unrealengine.com/udk/Two/WebHome.html)
Unreal Engine 4 (EPIC, https://www.unrealengine.com/what-is-unreal-engine-4)
VRML/X3D (http://www.web3d.org/x3d/specifications)
Vulkan (Khronos Group, https://www.khronos.org/vulkan/)

Autorinnen und Autoren

von Mammen, Sebastian, leitet die Arbeitsgruppe Games Engineering am Lehrstuhl für Mensch-Computer-Interkation an der JMU. Sein wissenschaftlicher Fokus liegt auf der Modellierung und interaktiven Simulation komplexer Systeme.

Knote, Andreas, hat Mathematik (B.Sc.) und Informatik (M.Sc.) an der Universität Augsburg studiert und promoviert nun im Bereich Games Engineering mit einem Forschungsschwerpunkt im Bereich der Modellierung und interaktiven Simulation entwicklungsbiologischer Systeme.

Roth, Daniel, hat Medien- und Bildtechnik (M.Eng.) an der Hochschule Köln studiert und ist wissenschaftlicher Mitarbeiter am Lehrstuhl für Mensch-Computer-Systeme an der JMU und promoviert im Bereich Human-Computer-Interaction zum Thema interpersonelle Synchronisation.

Latoschik, Marc Erich, leitet den Lehrstuhl für Mensch-Computer-Interaktion an der JMU. Er arbeitet seit über 20 Jahren an neuen, multimodalen und perzeptuellen Schnittstellen für Virtual, Augmented und Mixed Reality.