Das *parimod*-System

(Parallele Computergraphik und Animation mit Transputern)

K. Zeppenfeld, C. Landwehr, F. Thiesing, O. Vornberger
Fachbereich Mathematik/Informatik
Universität Osnabrück
49069 Osnabrück
Germany
klaus@informatik.Uni-Osnabrueck.DE

Abstract. Das *parimod*-System, welches als Abkürzung für "The parallel computer graphics and interactive solid modeling system" steht, verbindet in bisher einzigartiger Weise die Gebiete der parallelen Algorithmen bzw. Multiprozessor-Systeme und der Computergraphik miteinander. Es besteht aus einzelnen Modulen, die seit 1991 an der Universität Osnabrück entwickelt wurden. Intention dieser Entwicklung war es, die Flexibilität und die universelle Einsetzbarkeit von Transputer-Systemen anhand verschiedener Anwendungen aus der Computergraphik zu verdeutlichen. Entstanden ist dabei ein transputerbasiertes graphisches System zur schnellen und interaktiven Erstellung, Berechnung und Anzeige bzw. Animation von dreidimensionalen Szenen.

1. Einleitung

In den letzten Jahren haben zwei Gebiete der Informatik, die der parallelen Algorithmen auf Multiprozessor-Systemen und der Computergraphik einen rasanten Entwicklungsschub erfahren. Durch das Zusammenschalten von vielen einzelnen Universalprozessoren wird dem Benutzer hohe Rechenleistung, große Flexibilität und eine an der Probleminstanz skalierbare Hardware geboten. Das Vorhaben, fotorealistische Bilder einzig und allein durch einen Computer erzeugen zu lassen, erfordert aber genau diese hohe Rechenleistung, Flexibilität und Skalierbarkeit von Multiprozessor-Systemen. Gesellt sich zur Berechnung einzelner Bilder auch noch der Wunsch ganze Animationssequenzen oder virtuelle Welten in Echtzeit zu erzeugen und zu durchwandern, kann dieser enorme Rechenzeitbedarf nur noch durch Parallelrechner oder durch Rechner mit Spezialhardware zur Durchführung graphischer Berechnungen annähernd gedeckt werden.
Was liegt also näher, als diese beiden Gebiete der Informatik miteinander zu verknüpfen und zu untersuchen, wie sich die Standardalgorithmen der Computergraphik (vgl. [FDFH90] oder [WaWa92]) parallelisieren lassen, um somit zu einer schnelleren Berechnung beizutragen. Dabei wird hier nun der Versuch unternommen diese benötigten Geschwindigkeitssteigerungen einzig und allein durch algorithmische Lösungen zu erzielen im Gegensatz zum Ansatz bei der Verwendung von Spezialhardware.
Damit die so erzielten Ergebnisse aber auch wirkungsvoll und einfach vom Benutzer angewendet werden können, sind diese verschiedenen Algorithmen als Module zu einem Gesamtsystem namens *parimod* (The parallel computer graphics and interactive solid modeling system) zusammengefaßt. Es umfaßt von der interaktiven graphischen Szenenmodellierung bis hin zur

Darstellung und Animation ein weites Spektrum der Computergraphik, implementiert in Form von parallelen Algorithmen auf Transputer-Systemen.
Das *parimod* -System besteht im wesentlichen aus zwei Komponenten. Zum einen sind das C-Programme und X-Applikationen ([KeRi83] , [ORe90a/b]), die auf der UNIX-Seite des Systems laufen ([BaRu84]), und zum anderen sind es parallele Algorithmen, die auf einem Transputer-System ablaufen ([INM88c], [INM89]) und in occam2 programmiert sind ([INM88a], [JoGo88]).
Die C-Programme und X-Applikationen dienen zur benutzerfreundlichen, interaktiven Szenenmodellierung, zur Bildanzeige auf X-Terminals und zum Verwalten der Datenströme zwischen der UNIX-Seite und den Transputern. Die parallelen Algorithmen führen die schnelle Berechnung und Anzeige der modellierten Szenen als Einzelbilder oder animierte Bildsequenz auf einem Multiprozessor-System, bestehend aus 64 T800 Prozessoren, durch. Die berechneten Szenen können wahlweise auf einem X-Terminal mit bis zu 256 verschiedenen Farben angezeigt werden, oder direkt auf dem Graphical Display System (GDS) im True-Color-Modus, welches direkt ins Transputer-System integrierbar ist (vgl. [Par90b]).
Abbildung 1 zeigt einen schematischen Überblick des *parimod*-Systems und seiner Einzelkomponenten, die in den nächsten Kapiteln im Detail beschrieben werden.

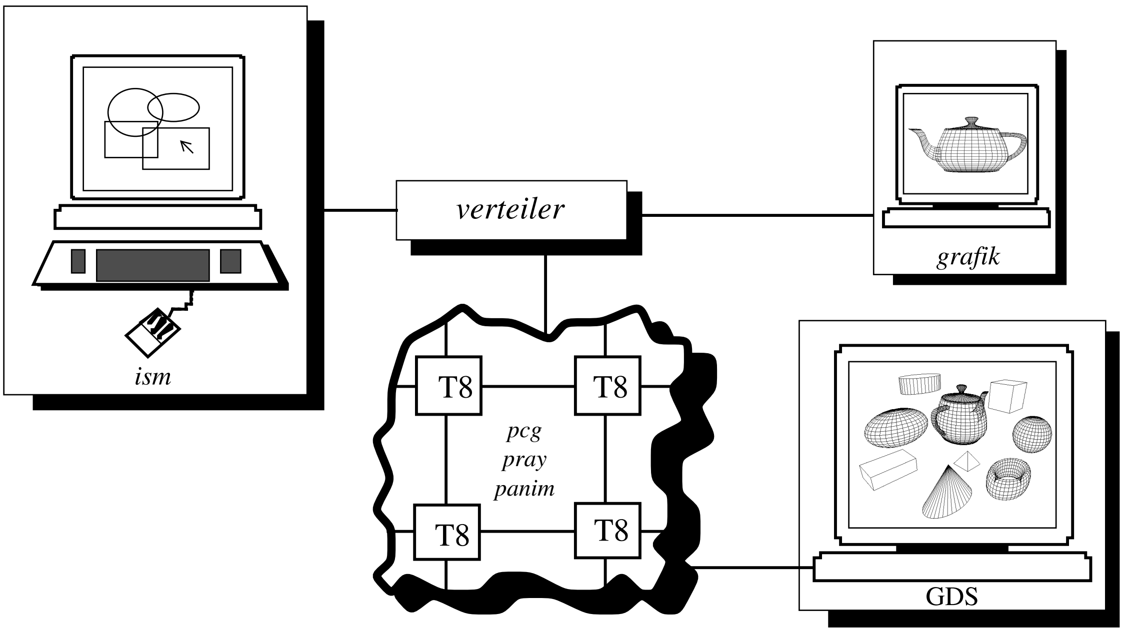

Abbildung 1: Das *parimod* - System

Der Rest dieses Artikels gliedert sich wie folgt: Im nachfolgenden Kapitel wird zunächst der interaktive graphische Szenenmodellierer *ism* beschrieben, mit dem auf der UNIX-Seite des *parimod*-Systems Szenenbeschreibungen generiert werden können. Das C-Programm *verteiler* lenkt die Datenströme von der UNIX-Seite zu den Transputern und leitet umgekehrt die berechneten Bilder zur X-Applikation *grafik*, mit der die Ergebnisse dann monochrom oder mit bis zu 256 verschiedenen Farben dargestellt werden können. Die Kommunikation zwischen der UNIX-Seite und dem Transputer-System wird vom Programm *verteiler* über den S-Bus der Workstation, an der das Transputer-System angeschlossen ist, geregelt. Eine detaillierte Beschreibung dieser beiden Programme würde aber den Rahmen dieses Artikels sprengen, so daß hier nur einfach ihre Funktionalität beschrieben wird.
Das Kapitel 3 beschäftigt sich dann ausführlich mit den Transputer-Programmen *pcg*, *pray*, und *panim*, die die parallele Berechnung zur Darstellung der Szenen in unterschiedlicher Verfahren durchführen und sogar Animationen von Bewegungsabläufen durch einzelne Szenen ermöglichen.
Im Kapitel 4 schließt eine Zusammenfassung und Ausblick über weitere Forschungen auf diesem Gebiet den Artikel ab.

2. Der interaktive Szenenmodellierer *ism*

Zur Konstruktion von dreidimensionalen Szenen bietet das *parimod*-System die Möglichkeit den interaktiven Szenenmodellierer *ism* (interactive solid modeler) zu verwenden. Diese Applikation ist in der Programmiersprache C ([KeRi83]) geschrieben und verwendet als Benutzeroberfläche das X-Window-System (vgl. [ORei90a/b]). Durch die Verwendung dieses Window-Systems besteht die Möglichkeit Anwenderprogramme, wie z. B. *ism* selbst, remote auf einem beliebigen Rechner im Netzwerk ablaufen zu lassen, während die Ergebnisse auf der lokalen Workstation angezeigt werden.

Die Entwicklung von *ism* basiert auf dem Wunsch nicht nur Szenen durch den Computer schnell zu berechnen, sondern auch ebenso einfach konstruieren und modifizieren zu können, um somit ein einfaches Werkzeug zur Manipulation von Szenen zur Verfügung zu haben.

Hauptproblem ist dabei aber die Frage, wie eine dreidimensionale Szene auf einem zweidimensionalen Bildschirm dargestellt werden kann, so daß der Benutzer einen Eindruck vom Aussehen der Szene bekommt. Dieses Problem wird in *ism* gelöst, wie es auch im Bereich der technischen Zeichnungen üblich ist. Eine Szene wird nämlich einfach in ihren Grund- und Aufriß aufgeteilt.

Somit kann *ism* auch als elektronisches Zeichenbrett gesehen werden, mit dem der Benutzer eine aus einzelnen Objekten zusammengesetzte Szene modellieren kann. In Abbildung 2 ist die graphische Oberfläche von *ism* während der Konstruktion einer Beispielszene dargestellt.

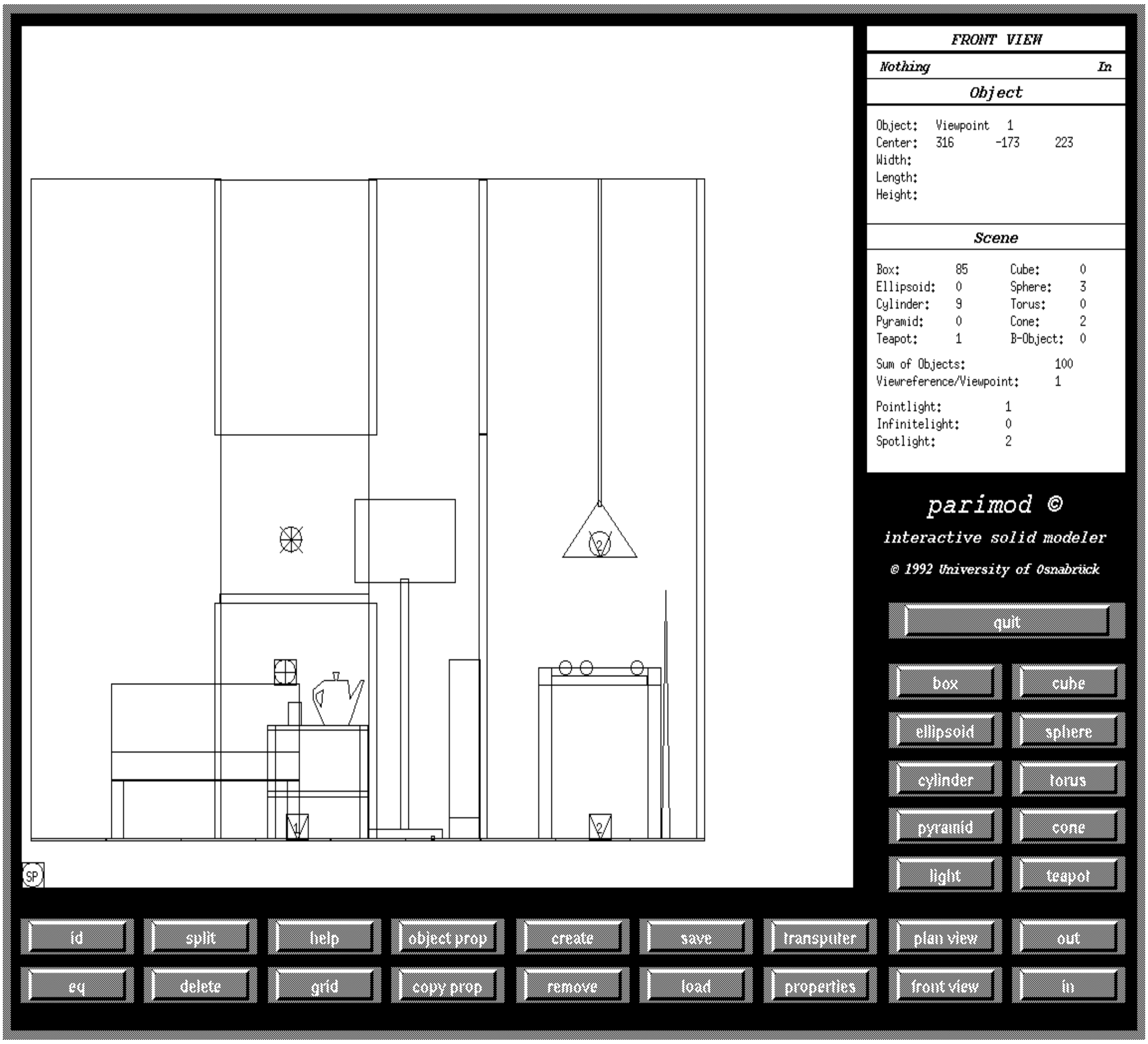

Abbildung 2: *ism*

Im wesentlichen teilt sich die Applikation in drei Teilbereiche auf. In der oberen rechten Ecke werden alle Informationen zur momentan behandelten Szene auf einen Blick dargestellt. Der Benutzer sieht dort, ob er sich im Grund- oder Aufriß der Szene befindet, welches aktuelle Objekt gerade ausgewählt wurde, bzw. welche Lage und Position es zur Zeit in der Szene hat. Ebenso kann abgelesen werden, aus wievielen Objekten die gesamte Szene zur Zeit besteht.
Damit fällt der Blick automatisch auf den zweiten Hauptteil von *ism*, den Knöpfen, die sich auf der rechten unteren und auf der gesamten unteren Seite der Applikation befinden. Im bisher noch nicht erwähnten linken oberen Teil befindet sich die Zeichenfläche, in der jeweils Grund- oder Aufriß der Szene dargestellt wird, bzw. in der die Szene modelliert werden kann.
Mit Hilfe der Knöpfe und der Maus wird aber nun der eigentliche Teil der Konstruktion erledigt. Auf der rechten Seite befinden sich unterhalb des quit-Knopfes, durch den die Applikation beendet werden kann, die Basisobjekte, aus denen die Szenen zusammengesetzt werden können. Neben Quader, Würfel, Kugel und Zylinder gehören auch Ellipsoid, Torus, Pyramide, Kegel und der Utah Teapot (vgl. [Cro87]) zum Lieferumfang der Basisobjekte von *ism*, die nach dem Baukastenprinzip zu einer Szene zusammengesetzt werden können. Dies geschieht einfach durch Anklicken des jeweiligen Objektknopfes und nachfolgendem Aufziehen des Objekts auf die gewünschte Größe im Zeichenfeld.
Alle Objekte sind mit sogenannten *Henkeln* (vgl. [Mänt88]) versehen, die unsichtbar an den Umrissen angebracht sind. Wird ein solcher Henkel mit der Maus getroffen, so ist das Objekt auf der Zeichenfläche verschiebbar oder kann in seiner Größe verändert werden.
Die Darstellung des Objekts während des Verschiebens bzw. der Größenveränderung geschieht mit sogenannten *Rubberband*-Techniken (vgl. [BoGi82], [Mänt88] und [Mor85]), bei denen das Objekt während der Veränderung abwechselnd gelöscht und neu gezeichnet wird. Liegen mehrere Objekte übereinander, so kann durch mehrmaliges Drücken der Maus eines dieser Objekte ausgewählt werden, um es zu manipulieren. Welches Objekt gerade manipulierbar ist, wird anschaulich durch gestricheltes Hervorheben der Objektumrandung dargestellt und kann auch im Übersichtsfenster in der rechten oberen Ecke gesehen werden.
Der Konstruktionsbeginn findet immer im Grundriß statt, so daß hier bereits die x- und y-Längen der Objekte bestimmt werden. Den Wert der z-Koordinate und die Höhe eines Objekts im Raum werden dann im Aufriß bestimmt.
Genauso wie die Plazierung von Objekten wird auch die Plazierung von Lichtquellen vorgenommen. Dabei hat der Benutzer die Auswahl aus Punktlicht, unendlichem Licht und Spots. Alle Lichtquellen verhalten sich nach dem durch PHIGS PLUS in [HHHW91] beschriebenem Standard. Das ambiente Licht, welches die Gesamthelligkeit der Szene beschreibt, wird bei der Definition der Szeneneigenschaften definiert.
Wird zur Definition der Szeneneigenschaften der properties-Knopf angewählt, erscheint das Menü, welches in Abbildung 3 dargestellt ist.
Hier kann der Benutzer entweder durch Anklicken von Knöpfen oder durch das Aufziehen sogenannter *Slider* (z.B. red,

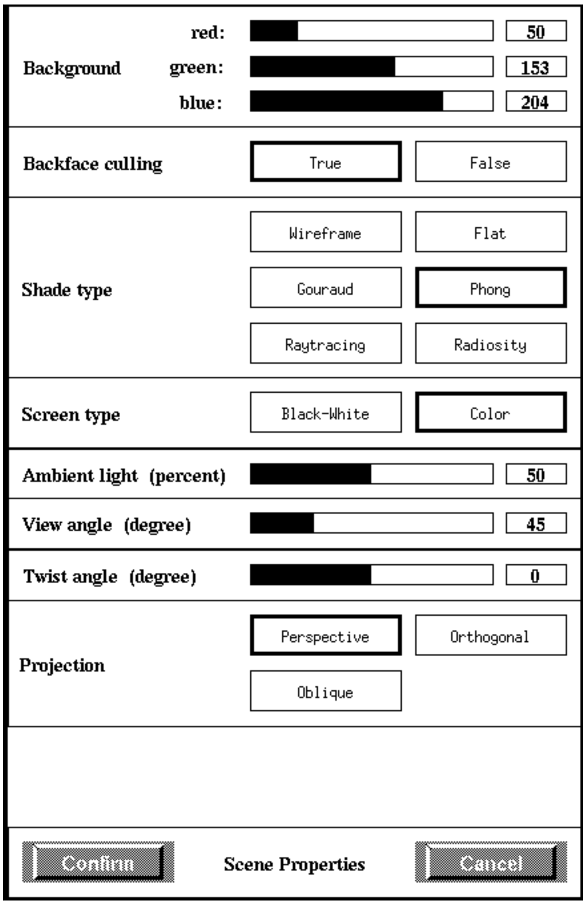

Abbildung 3: Szeneneigenschaften

green oder blue in Abbildung 3 oben) einfach mit der Maus Eigenschaften wie etwa Hintergrundfarbe, Shading-Art, Neigungswinkel der Kamera, Projektionsart u.v.m. bestimmen.
Da auch für jedes Objekt bzw. für jede Lichtquelle eigene Eigenschaften zu definieren sind, erscheint bei Anklicken des objectprop-Knopfes ein ähnliches Menü, welches Abbildung 4 zeigt.

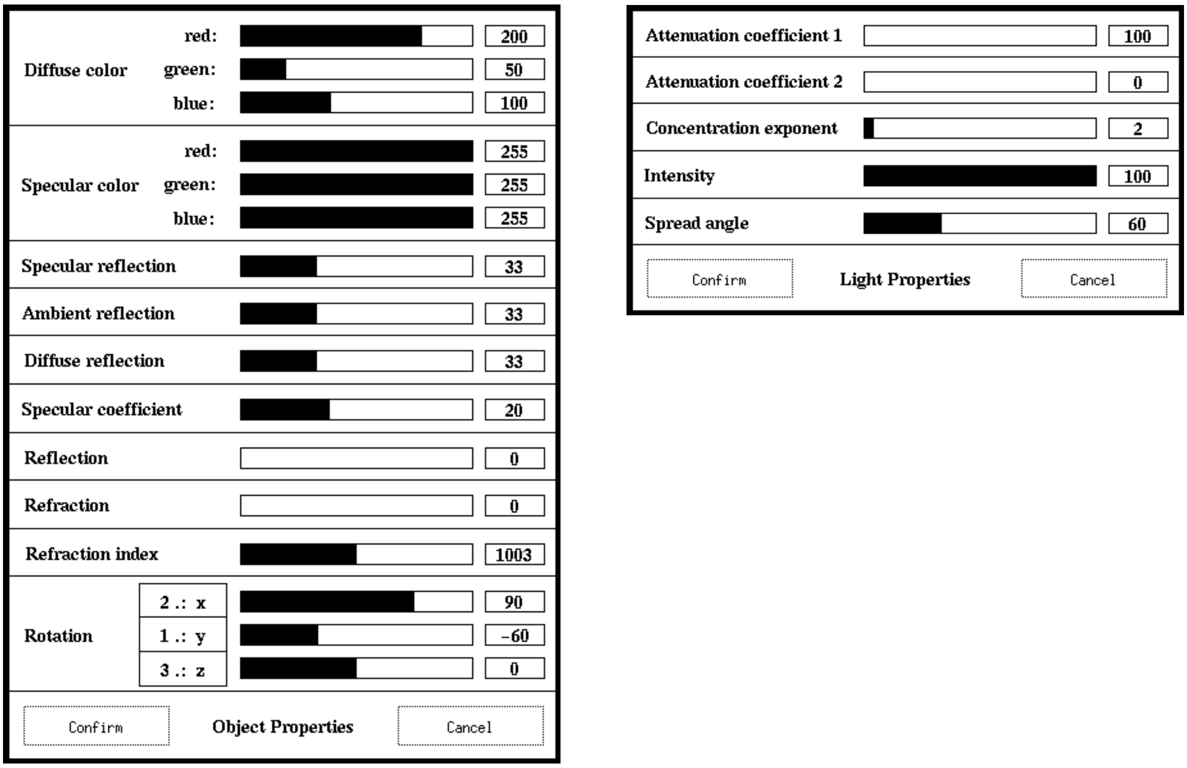

Abbildung 4: Objekt- und Lichteigenschaften

Hier kann nun für jedes einzelne Objekt die Farbe, Durchsichtigkeit, rotierte Position im Raum etc. bestimmt werden und für jede Lichtquelle die Intensität oder beispielsweise der Abstrahlwinkel eines Spotlichts.
Ist ein Objekt mit all seinen Eigenschaften definiert, so können diese auch einfach mit Hilfe des copyprop-Knopfes auf andere Objekte kopiert werden.
Die Eigenschaften, die mit den Menüs aus den Abbildungen 3 und 4 eingestellt werden können, enthalten die wesentlichen Auswahlmöglichkeiten, mit denen Objekte definiert werden können, und wie sie für die zur Zeit gängigen Schattierungsalgorithmen in der Computergraphik benötigt werden (vgl. [FDFH90]). Dadurch hat der Benutzer einen großen Spielraum, mit dem er ohne viel Aufwand verschiedene Szenenvariationen erstellen kann.
Damit nicht immer alle Eigenschaften für jedes Objekt bzw. für jede Szene von Grund auf neu definiert werden müssen, initialisiert *ism* automatisch diese Eigenschaften mit wohldefinierten Werten vor, so daß der Anwender auch ohne jeweils vollständige Definition der Eigenschaften vernünftige Bilder zu sehen bekommt.
Mit Hilfe der sogenannten *Splitting Plane* bietet *ism* die Möglichkeit die Basisobjekte aufzuteilen, damit auch Objekte, die nicht aus Basisobjekten aufgebaut sind, erzeugt werden können. Durch die Definition der Orientierung einer Ebene im Raum mit Hilfe des Normalenvektors wird ein Objekt an dieser Ebene durch Drücken des split-Knopfes in zwei neue Objekte unterteilt. Diese können dann zur weiteren Modellierung in der Szene verwendet werden. Die grundlegende Idee der Splitting Plane ist aus [Mänt88], und [Mor85] entnommen und dient als erster Schritt der Implementierung von boole'schen Operatoren (Vereinigung, Durchschnitt, Komplement) zur universellen Szenenmodellierung.
Da die Implementierung dieser Operatoren nicht trivial ist, wurde dies in *ism* zunächst in Form

der Splitting Plane verwirklicht, so daß alle Optionen für eine Erweiterung noch offen sind (vgl. [Mänt88]).
Sollen Lichtquellen, die Splitting Plane oder der Betrachterstandpunkt, welcher beim Beginn einer neuen Szenenmodellierung automatisch definiert wird, außerhalb der Zeichenfläche positioniert werden, so bietet *ism* die Möglichkeit, durch die Verwendung des out-Knopfes die Szene von außerhalb zu betrachten. Dies ist sehr hilfreich, da die Objekte die gesamte Zeichenfläche einnehmen können, und z.B. der Betrachterstandpunkt dann weiter außerhalb liegen muß, wenn die Szene als Ganzes und ohne Verzerrungen betrachtet werden soll.
Ist *ism* mit einem Transputer-System verbunden (siehe Abbildung 1), so kann eine gerade modellierte Szene on line an das Transputer-System geschickt werden (transputer-Knopf) und dort mit den im weiteren Verlauf dieses Artikels beschriebenen parallelen Programmen schattiert werden. Wird das Programm *pcg* verwendet, so entsteht, je nachdem welche Art der Darstellung ausgewählt wurde, innerhalb weniger Sekunden die mit *ism* beschriebene Szene auf dem Bildschirm, und der Betrachter kann sofort entscheiden, ob die gerade modellierte Szene seinen Vorstellungen entspricht. Nach jeder Veränderung kann die Szenenbeschreibung sofort wieder in das Transputer-Netzwerk gesendet werden und von dort aus erneut dargestellt werden.
Unabhängig davon dienen die save- und load-Knöpfe zum Abspeichern bzw. Laden von Szenenbeschreibungsdateien, die mit *ism* erstellt worden sind. Diese Szenenbeschreibungen sind ASCII-Dateien, in denen alle Eigenschaften einer Szene im sogenannten *parimod*-File-Format abgespeichert sind. Damit das parimod-File-Format auch von anderen Graphikprogrammen als Front-End nutzbar ist bzw. auch noch erweitert werden kann, existiert eine vollständige Beschreibung der Grammatik in Backus-Naur-Form (vgl. dazu auch [Zep93]).
Nicht nur aufgrund des parimod-File-Formats und der zugehörigen Grammatik, sondern auch wegen der modularen Programmierung des C-Programms bzw. der X-Applikation ist *ism* so konzipiert, daß Erweiterungen bzw. Anpassungen an andere Graphiksysteme einfach möglich sind.
Insgesamt ist *ism* als Front-End zur Beschreibung von Szenen ein hilfreiches und benutzerfreundliches Werkzeug, das gerade durch sein interaktives Konzept in Verbindung mit dem angeschlossenen Transputer-System das Erstellen, Modifizieren und Betrachten von dreidimensionalen Szenen stark vereinfacht.

3. Parallele Computergraphik

In diesem Kapitel werden nun die Programme *pcg*, *pray*, und *panim* beschrieben, die die parallele Berechnung der durch *ism* erstellten Szenenbeschreibungen in unterschiedlicher Weise durchführen (vgl. Abbildung 1).

3.1 Das parallele Computergraphik-Tool *pcg*

Das *pcg*-Programm (parallel computer graphics tool) ist eine Parallelisierung der klassischen Viewing-Pipeline, bei der zur Darstellung der Szene das z-Buffer-Verfahren verwendet wird (vgl. [FDFH90] und [WaWa92]). Mit *pcg* können Szenen, die mit *ism* erstellt worden sind, parallel schattiert und dargestellt werden. Die Darstellung der Szenen kann dabei wahlweise auf dem GDS-Monitor des Transputer-Systems oder durch die X-Applikation *grafik* auf einem X-Terminal angezeigt werden (vgl. Abbildung 1).
Die verschiedenen Darstellungsarten, mit denen Szenen durch *pcg* dargestellt werden können, beginnen bei einfacher Liniendarstellung und gehen über Flat Shading, Gouraud Shading bis hin zum qualitativ hochwertigen Phong Shading (vgl. [FDFH90]). Die Unterschiede der einzelnen Shadingverfahren sind sehr deutlich zu erkennen, wenn die Ausgabe auf dem ins Transputer-System integrierbaren GDS-Monitor im True-Color-Modus erscheint. Wird anstatt dieses Monitors ein X-Terminal zusammen mit der X-Applikation *grafik* verwendet, so wird unabhängig vom Bildschirmtyp und davon, ob die Bildinformation in Form von RGB-Farbwerten oder komprimiert schwarz-weiß vorliegt, zunächst ein Schwarz-Weiß-Bild mit Hilfe

von Dithering-Verfahren (vgl. ebenfalls [FDFH90] und [WaWa92]) angezeigt. Nach dem Bildaufbau kann die Bildinformation abgespeichert werden, wobei das Schwarz-Weiß-Bild als X-Bitmap und das Farbbild als X-Pixmap im X-Window-Dump-Format (vgl. [ORei90a]) abgespeichert werden. Beide X-Maps eignen sich aufgrund ihres genormten Formats zur Weiterverarbeitung. Auf Farbbildschirmen kann per Knopfdruck aus dem X-Pixmap ein Farbbild generiert und im selben Fenster angezeigt werden. Die am Ende dieses Artikels abgebildeten Szenen in Abbildung 12 und 13 sind auf die hier beschriebene Weise entstanden.

3.1.1 Parallelisierung durch Bildraumaufteilung

Durch umfangreiche Meßreihen, in denen mit der sequentiellen Version der Viewing-Pipeline diverse Testbilder berechnet wurden, zeigte sich, daß der größte Rechenzeitbedarf auf das Schattieren der Flächen entfällt. Insbesondere beim qualitativ hochwertigen Phong Shading, bei dem jedes Pixel einzeln beleuchtet wird, beträgt der Anteil für das Shading nahezu 95%. Der Aufwand zur Berechnung schattierter Bilder mit diesen Verfahren ist also in hohem Maß proportional zur Anzahl der Pixel. Dies rechtfertigt als Idee zur Parallelisierung die Aufteilung des Bildschirms in viele kleine Teilfenster. Zugleich wird dadurch auch der Notwendigkeit zur Aufteilung des z-Buffers und des Bildschirmspeichers Rechnung getragen, die wegen ihres großen Speicherbedarfs nur für kleine Teilfenster auf den im Netzwerk verfügbaren 1- und 2MByte-Transputern gehalten werden können. Schematisch wird diese Idee der Parallelisierung, die allgemein auch als Bildraumaufteilung bezeichnet wird, in Abbildung 5 dargestellt.

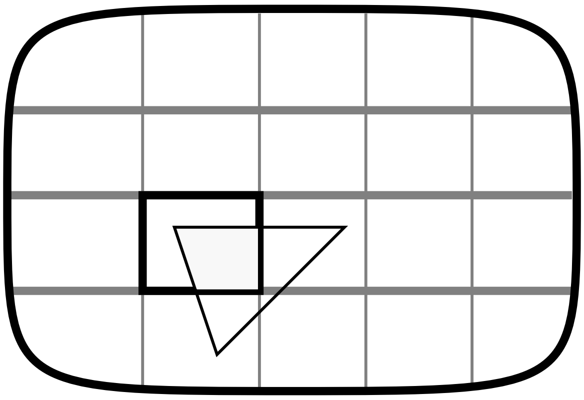

Abbildung 5: Bildraumaufteilung

Die Anzahl der Teilfenster bei der in Abbildung 5 dargestellten Parallelisierungsstrategie sollte dabei die der eingesetzten Transputer übersteigen, um eine gute Lastverteilung zu ermöglichen, da Szenen nicht notwendig gleichmäßig über das ganze Bild verteilt sind. Teilbilder, in die keine Objekte fallen, lassen sich erheblich schneller bearbeiten als solche, in die viele Objekte abgebildet werden. Das Problem hierbei ist die Entwicklung einer Heuristik zur gleichmäßigen Lastverteilung, die entscheidet, wieviele und welche der Teilbilder von einem bestimmten Transputer im Netz zu berechnen sind. Außerdem ist die Zahl und Größe der Teilbilder zu optimieren.

Damit ist die Idee für eine erste Parallelisierung vorgegeben, die im wesentlichen darin besteht, den sequentiellen Algorithmus auf jedem Netzwerk-Transputer ablaufen zu lassen, so daß zur gleichen Zeit an so vielen Teilbildern gearbeitet wird, wie sich Transputer im Netzwerk befinden. Dabei ist aber immer noch das Problem zu lösen, wie ein Transputer sein eindeutiges Teilproblem erhält und wie das fertige Teilbild, also ein Teil des Bildschirmspeichers, zur Ausgabe gelangt.

Der Algorithmus, der in dieser ersten Parallelisierung auf den Prozessoren abläuft, läßt sich wie folgt beschreiben: Nachdem die Transputer mit dem Programm geladen worden sind, läuft auf jedem zunächst ein initialer Prozeß. Dieser empfängt und speichert die notwendige Szeneninformation für das zu berechnende Gesamtbild, die vom Host-Transputer in das Netzwerk geschickt wird. Jeder Transputer speichert die Szeneninformation und schickt sie an seinen Nachbarn weiter. Danach startet er unter anderem den Rendering-Prozeß und wartet darauf, daß ihm ein Teilproblem zugeteilt wird. Dies geschieht durch eine im Netzwerk umlaufende Problemmeldung, die das nächste noch zu bearbeitende Teilfenster spezifiziert. Dafür sind vier Werte ausreichend, nämlich die (x, y)-Werte der linken oberen Ecke in Pixelkoordinaten und

die Breite und Höhe des Teilfensters in Pixeln. Die initiale Problemmeldung mit dem Teilfenster links oben wird vom Host-Transputer in das Netzwerk geschickt. Ein Transputer, der gerade kein Teilbild berechnet, speichert das Teilproblem und setzt die Problemmeldung auf das nächste Teilfenster, bevor er sie an seinen Nachbarn weiterreicht und mit der Berechnung seines Teilbildes beginnt. Ein beschäftigter Prozessor gibt die Meldung unverändert weiter. Dadurch ergibt sich eine dynamische Lastverteilung, da nur freie Transputer ein Teilproblem bekommen, solange noch welche zu vergeben sind. Der Prozessor, der das letzte Teilfenster zur Berechnung erhalten hat, schickt eine Meldung an den ausgezeichneten Transputer, der als einziger im Netz mit dem Host-Transputer verbunden ist. Dieser leitet dann die Terminierung ein, die nach den üblichen Methoden für verteilte Algorithmen durchgeführt wird. Verwendet werden kann dafür z.B. der Echo-Algorithmus oder die Methode des Verschickens von Farbmeldungen in Wellen über alle Prozessoren (vgl. [Leig92]).
Ein berechnetes Teilbild muß vom Transputer zunächst in Richtung Workstation oder GDS verschickt worden sein, bevor er das nächste Teilproblem bearbeiten kann, denn erst danach sind z-Buffer und Bildschirmspeicher wieder frei zur Aufnahme eines neuen Teilbildes. Jeder Netzwerk-Transputer muß also zusätzlich in der Lage sein, ein Teilbild von seinem Nachbarn zu empfangen und es auf dem günstigsten Weg zum Host-Transputer bzw. GDS-Transputer zu versenden.
Der Algorithmus auf einem der Netzwerk-Transputer lautet dann schematisch wie folgt:

```
empfange Problemmeldung (n)
falls Rendering-Prozeß frei
    setze Problemmeldung (n) auf Teilfenster (n+1)
    falls Teilfenster (n+1) existiert
        schicke Problemmeldung (n+1) an Nachbarn
        Rendering-Prozeß nicht frei
        berechne Teilbild (n)
        schicke Teilbild (n) an Host-Transputer
        Rendering-Prozeß frei
    sonst
        schicke Meldung "fertig" an Transputer 0
sonst
    schicke Problemmeldung (n) an Nachbarn
```

Für die umlaufende Problemmeldung und evtl. auch für die verwendete Terminierung wird im Netzwerk ein Hamilton-Kreis benötigt. Graphentheoretisch handelt es sich dabei um einen geschlossenen Weg, der alle Knoten genau einmal besucht. Wenn die Problemmeldung auf dem Hamilton-Kreis umläuft, wird sichergestellt, daß alle Prozessoren gleich häufig mit ihr konfrontiert werden. Die fertigen Teilbilder sollten auf dem kürzesten Weg zum Host-Transputer bzw. GDS-Transputer gelangen, um eine schnelle Ausgabe zu gewährleisten und möglichst wenig "Zwischen"-Transputer mit dem Weiterleiten dieser Teilbilder zu belasten.
Als Topologie, in der alle diese gewünschten Eigenschaften vereinigt sind, wird ein *deBruijn*-Netzwerk verwendet. Diese Netzwerke eignen sich nämlich aus vielen Gründen hervorragend zur Abbildung von Transputer-Topologien (vgl. [Leig92]).
Weitgehende Messungen ergaben, daß ein Minimum der Rechenzeit erreicht wird, wenn die Zahl der Teilbilder das Drei- bis Fünffache der Zahl der Transputer beträgt. Die Problemmeldung läuft etwa 15 mal pro Sekunde im Netzwerk um. Dadurch hält sich die Wartezeit der Rendering-Prozesse stark in Grenzen, ohne daß zu häufiges Weiterleiten der Problemmeldung den Transputer zu sehr belastet, so daß die Prozessorzeit für den Rendering-Prozeß stark sinkt.
Die Parallelisierung zeichnet sich durch die dynamische Lastverteilung und die sofortige Teilbildausgabe aus, die es dem Betrachter erlaubt, den Bildaufbau mitzuverfolgen. Die kurze Antwortzeit von wenigen Sekunden, also die Zeit vom Start der Berechnungen bis zur Ausgabe

des ersten Teilbildes, verkürzt subjektiv die Wartezeit auf das ganze Bild, das sich danach aus bis zu drei Teilbildern pro Sekunde zusammensetzt. Dabei wird die Ausgabe, wenn sie auf dem X-Terminal erfolgt, durch die relativ geringe Übertragungsrate auf dem Bus erheblich gebremst. Bei der Ausgabe auf dem GDS-Monitor ist nahezu keine Verzögerung festzustellen, so daß die fertigen Teilbilder sofort erscheinen.

Zu klären bleibt der Einfluß der mehrfachen Objekterzeugung. Jeder Transputer erzeugt für jedes Teilbild die komplette Szene. Zwar verringert sich durch evtl. durchgeführtes Clipping (vgl. [FDFH90]) die Zahl der Polygone, die in Betracht gezogen werden müssen, aber trotzdem bleibt ein sequentieller Anteil im parallelen Algorithmus, der den Speedup begrenzt.

3.1.2 Parallelisierung durch Objektraumaufteilung

Die zweite Möglichkeit der Parallelisierung ist eine Weiterentwicklung der ersten mit dem Ziel, die Objektgenerierung zu zentralisieren und damit das überflüssige mehrfache Erzeugen der Objekte für jedes Teilfenster zu vermeiden. Von der ersten Parallelisierung werden die Topologie und, wegen des Speicherplatzproblems, die Aufteilung des Bildes in Teilfenster übernommen. Für die zentrale Objekterzeugung ist ein gesonderter Transputer zuständig (in Abbildung 6 als Wurzel des Baumes sichtbar), dessen Programm sich von dem der Rendering-Transputer unterscheidet. Dazu muß das sequentielle Programm unter den Transputern aufgeteilt werden, wodurch die Viewing Pipeline verteilt abläuft.

Die Notwendigkeit, z-Buffer und Bildschirmspeicher disjunkt aufzuteilen, verhindert eine Parallelisierung, bei der die einzelnen Objekte zum Rendering einzelnen Prozessoren übergeben werden. Denn im allgemeinen wird ein Objekt über die Grenzen eines Teilbildes hinausragen und damit in den Bildschirmspeicherbereich eines anderen Transputers fallen, der dieses Objekt aber nicht erzeugt. Außerdem ist diese Art der Parallelisierung zu grob, da die Zahl der Objekte im Verhältnis zur Prozessoranzahl relativ klein ist und die einzelnen Objekte je nach Art und Größe im Aufwand für ihre Darstellung stark differieren.

Die genannten Probleme lassen sich lösen, wenn statt der kompletten Objekte ihre Bestandteile im Netzwerk verteilt werden. Spezialisierte Rendering-Prozessoren sorgen für die Darstellung von Linien bzw. Dreiecken, die von einem zentralen Erzeuger-Prozessor ins Netzwerk geschickt werden. Die Zahl der Linien oder Dreiecke, die im weiteren zusammenfassend als Atome bezeichnet werden, übersteigt die Zahl der Prozessoren um ein Vielfaches und ermöglicht so eine homogenere Verteilung unter den Rendering-Prozessen. Jeder Rendering-Transputer erhält eindeutige Bildschirmteile zugeteilt, in die er die Atome schattiert.

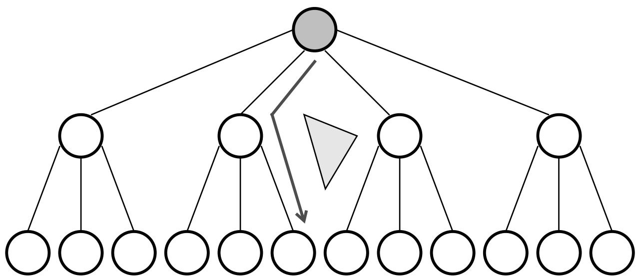

Abbildung 6: Objektraumaufteilung

Dadurch ist der Aufwand für jeden Prozessor in erster Linie von der Anzahl der Pixel in seinen Bildteilen abhängig. Eine gleichmäßige Bildaufteilung verspricht außerdem, zu einer guten Lastverteilung beizutragen. Schematisch ist diese Art der Parallelisierung, die auch als Objektraumaufteilung bezeichnet wird, in Abbildung 6 dargestellt.

Der Erzeuger-Prozessor benötigt die Information über die Bildaufteilung, denn er muß die Atome der erzeugten Objekte zu den Rendering-Transputern schicken, in deren Bildteile diese fallen. Wenn ein Atom die Fenstergrenzen überschreitet, also mehrere Prozessoren zur Darstellung benötigt, schickt der Erzeuger dieses Atom an alle betroffenen Transputer.

Zur Realisierung dieser Parallelisierung wird die Viewing Pipeline aufgespalten. Der Erzeuger generiert ein Objekt und verschickt dessen Atome nur, wenn es irgendwo im gesamten Bildschirmfenster sichtbar ist. Jeder Rendering-Transputer empfängt die für ihn bestimmten Atome und schattiert sie in seinem Bildschirmspeicher. Nach der vollständigen Berechnung treiben

die Prozessoren ihre Teilbilder aus. Aus dieser Idee ergibt sich die Frage nach der besten Aufteilung des Bildschirms, nach dem Format der Atome und nach der Ausgabe der berechneten Teilbilder.
Wie bei der ersten Parallelisierung wird zunächst auf jedem Transputer im Netz ein initialer Prozeß gestartet, der über den Hamilton-Kreis die Szeneninformation erhält. Zusätzlich wird jedem Transputer mitgeteilt, wie viele Rendering-Transputer (AnzProc) im Netzwerk vorhanden sind und wie viele Teilbilder (AnzPart) jeder davon berechnen soll. Daraus ermittelt jeder Netzwerk-Transputer nach demselben Algorithmus die Breite und die Höhe eines Teilbildes, wobei der Bildschirm in AnzProc $\times$ AnzPart gleich große Teilbilder aufgeteilt wird, die nur am rechten und unteren Rand kleiner sein dürfen. Die Teilbilder werden zeilenweise durchnumeriert, und jeder Rendering-Transputer merkt sich von AnzPart-vielen in einem Array die linke obere Ecke. Diese Teilbilder hat er zu berechnen. Damit die Aufteilung disjunkt geschieht, nimmt sich der Transputer mit der Nummer ProcId die Teilbilder mit der laufenden Nummer

ProcId + $i \cdot$ AnzProc, mit $i = 0, \ldots,$ AnzPart -1.

Abbildung 7 zeigt die Teilfenster (markiert) für Transputer 2 bei AnzProc = 4 und AnzPart = 5.
Das beschriebene Verfahren zur Vergabe der Teilbilder stellt zugleich die Heuristik zur Lastverteilung dar, denn alle Rendering-Prozessoren erhalten gleich große Bildausschnitte, die gleichmäßig über das Gesamtbild verteilt sind.

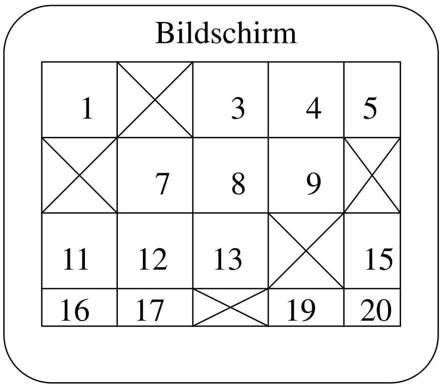

Abbildung 7: Teilfenster für Transputer 2

Nach dieser Initialisierungsphase startet jeder Transputer seine internen Prozesse. Durch die zentrale Objekterzeugung wird die mehrfache Generierung der Objekte vermieden. Bei geeigneter, nicht zu kleiner Teilbildgröße müssen lediglich relativ wenige Atome an mehrere Prozessoren geschickt werden. Der Bedarf an Kommunikation bei dieser Parallelisierung ist ausgesprochen hoch, da Tausende von Linien bzw. Dreiecke verschickt werden müssen. Aus diesem Grund müssen auf jedem Prozessor Puffer-Prozesse laufen, deren Kapazität zur Aufnahme von Linien- und Dreicksbeschreibungen ausreichen ist, um Deadlocks zu vermeiden.
Die Lastverteilung wird durch die beschriebene Heuristik zur festen Aufteilung der Teilbilder unter den Rendering-Prozessoren gewährleistet. Für gleichmäßig verteilte Szenen ist diese ausreichend gut. Ein Optimum ergibt sich für drei bis vier Teilbilder pro Prozessor. Feinere Aufteilungen mit dem Ziel, die Lastverteilung zu verbessern, führen dazu, daß immer mehr Atome zu mehreren Prozessoren geschickt werden müssen, wodurch der Kommunikationsaufwand überproportional steigt. Stark unausgewogene Szenen, bei denen sich die Objekte in wenigen Bildteilen ballen, führen zu einer schlechteren Lastverteilung.
Problematisch ist auch die unausgewogene Belastung der einzelnen Transputer bezüglich ihrer Kommunikation. Während die drei Rendering-Transputer, die direkt mit dem Erzeuger verbunden sind, wegen dieser Nachbarschaft alle Atome weiterleiten müssen, haben die Transputer, die weiter vom Erzeuger entfernt sind, diese Belastung nicht.

3.1.3 Ergebnisse und Vergleich

Subjektiv besteht der größte Unterschied zwischen den beiden vorgestellten Parallelisierungen darin, daß bei der ersten sukzessive die berechneten Teilbilder sofort ausgegeben werden, während der Benutzer bei der zweiten die komplette Berechnung abwarten muß, bevor die Szene auf dem Bildschirm erscheint. Da gerade im Rahmen des *parimod*-Systems interaktives Rendering der gewünschten Szene im Vordergrund steht, bietet sich hierfür die erste Parallelisierung besonders an.
Die algorithmischen Unterschiede betreffen vor allem den Speicherplatzbedarf, der bei der zweiten Parallelisierung deutlich größer ist, um eine zentrale Objekterzeugung zu ermögli-

chen. Im Gegensatz dazu beinhaltet die erste Parallelisierung mit ihrer mehrfachen Objekterzeugung offensichtlich Redundanz, die notgedrungen den Speedup beschränkt.
Um genauere qualitative Vergleiche zwischen den beiden Arten der Parallelisierung ziehen zu können, wurden fast hundert Beispielszenen zu Zeitmessungen herangezogen, um das Verhalten bei unterschiedlich komplexen Bildern und verschiedenen Shading- und Darstellungsarten zu ermitteln. Diese Messungen wurden auf bis zu 64 Prozessoren durchgeführt. Bei der Beurteilung von Szenen muß berücksichtigt werden, daß ihre Komplexität stark unterschiedlich sein kann. Dabei spielt nicht nur die Zahl und Art der beteiligten Objekte eine Rolle, sondern insbesondere auch ihre projizierte Größe auf dem Bildschirm und die Zahl der Lichtquellen. Je mehr Pixel zu berechnen sind, um so größer ist der Rechenaufwand, der in erster Linie beim Schattieren anfällt. Damit hat auch der Betrachterstandpunkt für die Rechenzeit eine unbestimmte, aber nicht zu unterschätzende Bedeutung. Der Einfluß der räumlichen Anordnung der Objekte wird durch die Aufteilung in Teilfenster noch verstärkt. Eine kleine Verschiebung eines Objekts oder des Blickwinkels kann dazu führen, daß für ein Objekt, das vorher nicht erzeugt werden mußte, nun Hunderte von Polygonen durch die Viewing Pipeline zu schicken sind.
Zusätzlich muß berücksichtigt werden, daß die drei Shading-Verfahren in ihrem Aufwand stark differieren. Das Phong Shading benötigt erheblich länger als das Gouraud Shading, obwohl es Szenen gibt, bei denen beide zu vergleichbaren Bildern führen. Es zeigt sich also, daß der Berechnungsaufwand stark szenenabhängig ist.
Deshalb mußte eine Auswahl von Szenen getroffen werden, die nur bedingt repräsentativ sein kann, aber trotzdem Rückschlüsse auf die Güte der Parallelisierung zuläßt. Um Vergleiche auch mit anderen Computergraphik-Programmen herstellen zu können, steht in zwei der anschließend diskutierten Szenen der Utah Teapot im Mittelpunkt der Betrachtung. Die Diskussion bezieht sich auf phong-schattierte Szenen, deren sequentielle Berechnung zwischen 170 und 710 Sekunden dauert. Obwohl die Szenen nur exemplarisch sein können, da anders als in anderen Bereichen der Informatik in der Computergraphik keine allgemeinen Benchmarks vorhanden sind, lassen sich doch gewisse Vor- und Nachteile der jeweiligen Parallelisierung anhand verschiedener Szenen erkennen. Dies wird an den folgenden exemplarisch ausgewählten Szenen deutlich.
Die erste Szene besteht aus einem Teapot, der über das gesamte Bild ragt (vgl. Abbildung 13 oben links). Bei ihrer Berechnung zeigt sich deutlich der Vorteil der zentralen, einmaligen Objekterzeugung. Der Speedup der zweiten Parallelisierung übertrifft den der ersten um das Anderthalbfache. Bei der ersten muß nämlich jeder Transputer den Teapot für jedes seiner Teilbilder, also etwa drei- bis viermal, erzeugen. Diese Redundanz entfällt bei der zentralen Objekterzeugung. Das umgekehrte Zeitverhalten ergibt sich für eine Szene mit 13 kleineren Teekannen, die über den ganzen Schirm verteilt sind (vgl. Abbildung 13 oben rechts). Dabei zeigt sich eindeutig der Effekt des Clipping, das jeden Transputer der ersten Parallelisierung nur etwa drei bis vier Kannen erzeugen läßt, während der zentrale Erzeuger der zweiten alle Kannen mit ihren insgesamt 35 187 Dreiecken generieren muß. Hierbei stellt also der Erzeuger-Prozessor den Flaschenhals des parallelen Algorithmus dar.
Die Vergleichbarkeit beider Parallelisierungen zeigt sich bei gemischten Szenen, z.B. von Innenräumen (vgl. Abbildung 12 und 13 unten), mit einigen Dutzend Objekten verschiedener Art, die relativ gleichmäßig über das Bild verteilt sind. Dabei erreichen beide Parallelisierungen mit 32 Transputern den gleichen mittleren Speedup von etwa 20. Dieses Ergebnis macht deutlich, daß die Heuristik zur statischen Verteilung der Teilbilder bei der zweiten Parallelisierung sich mit der dynamischen Verteilung der ersten messen kann.
Insgesamt läßt sich feststellen, daß beide Parallelisierungen die Rechendauer für qualitativ hochwertige, phong-schattierte Bilder von mehreren Minuten auf einige Sekunden reduzieren. Darstellungen mit Flat und Gouraud Shading oder sogar als Drahtmodell sind bereits nach noch kürzerer Zeit fertig berechnet. Der Speedup steigt mit wachsender Laufzeit des sequenti-

ellen Algorithmus sogar an, weil der Einfluß der Initialisierung des Netzwerks relativ zurückgeht. Daraus folgt der Effekt, daß die Laufzeit des parallelen Algorithmus nicht proportional mit der Komplexität der Szene ansteigt, und der Betrachter bereits nach vergleichsweise kurzer Zeit das fertige Bild sieht. Verkürzt wird die Wartezeit besonders durch die erste Parallelisierung, die jedes berechnete Teilbild sofort ausgibt und so den Betrachter den Bildaufbau mitverfolgen läßt. Insbesondere zeigte sich dies in der praktischen Anwendung bzw. während der Präsentation auf Ausstellungen. Dort wurde die erste Parallelisierung weitaus öfter verwendet, da sie sich auch didaktisch hervorragend zur Darstellung des Ablaufs einer parallelen Berechnung, eben im Sinne des *parimod*-Entwurfskonzepts, einsetzen läßt.

3.2 Der parallele Ray-Tracer *pray*

Der parallele Ray-Tracer *pray* (parallel ray tracer) wurde ebenfalls in occam2 implementiert und ist voll in das *parimod*-System integriert (vgl. Abbildung 1), d.h. *ism* kann zur Eingabebeschreibung und *grafik* als Ausgabe von *pray* genutzt werden. Im Gegensatz zu *pcg* werden durch *pray* die Szenen jetzt parallel nach dem Ray-Tracing-Verfahren berechnet (vgl. dazu [Gla89] oder [FDFH90]).

Im Unterschied zum bisher vorgestellten Verfahren der Schattierungsalgorithmen, die nach dem Prinzip der Viewing-Pipeline arbeiten, sieht ein Betrachter einen Punkt auf einem Objekt nicht mehr als das Resultat der Beziehung dieses Punkts bzw. der zugehörigen Fläche zu den direkten Lichtquellen, die von diesem Punkt aus sichtbar sind, sondern als eine Komposition von Strahlen anderer Flächen bzw. Lichtquellen der Szene, die sich in diesem Punkt vereinigen. In Abbildung 8 ist dieser Sachverhalt schematisch für den Punkt P dargestellt.

Da aber die Berechnung der einzelnen Objektpunkte, ausgehend von den Lichtquellen der Szene, viel zu aufwendig ist, wird beim Ray Tracing rückwärts vorgegangen. Die zugrundeliegende Idee besteht darin, anstatt von den Lichtquellen, vom Betrachterstandpunkt aus einen Sehstrahl durch die Mitte eines jeden Pixels der View Plane zu legen. Für diesen Strahl wird danach der Schnittpunkt mit dem ersten getroffenen Objekt bestimmt. Trifft der Strahl auf kein Objekt, erhält das Pixel die Hintergrundintensität. Ist das getroffene Objekt als spiegelnd charakterisiert, wird der Reflexionsstrahl weiter verfolgt. Bei einem transparenten Objekt wird zusätzlich noch der gebrochene Strahl weiter berechnet.

Daraus ergibt sich für die Szenenbeschreibung, daß nicht nur die Geometrie der Szene spezifiziert sein muß, sondern auch die optischen Eigenschaften der einzelnen Objekte gegeben sein müssen. Zur Berechnung von Schatten wird von jedem Schnittpunkt zwischen dem verfolgten Strahl und einem Objekt zu jeder Lichtquelle ein zusätzlicher Strahl ausgesandt. Trifft dieser Strahl auf ein blockierendes Objekt, dann liegt der Schnittpunkt im Schatten der entsprechenden Lichtquelle, und das von ihr ausgestrahlte Licht geht in die Berechnung der Intensität des Punkts nicht ein. Durch diese Beschreibung des Ray-Tracing-Verfahrens wird bereits deutlich, daß durch die Unabhängigkeit der Strahlen, die vom Auge aus durch die einzelnen Pixel gesendet werden, eine Aufteilung des Bildraums als eine einfache, aber leistungsstarke Möglichkeit der Parallelisierung angesehen werden kann, die schematisch in Abbildung 9 dargestellt wird.

Viele implementierte Systeme (vgl. [Gla89] oder [Gre91]) beschreiten diesen Weg, wobei sie sich dann aber in Details, wie etwa der Lastverteilung, unterscheiden. Letztendlich erzielt aber jede dieser Varian-

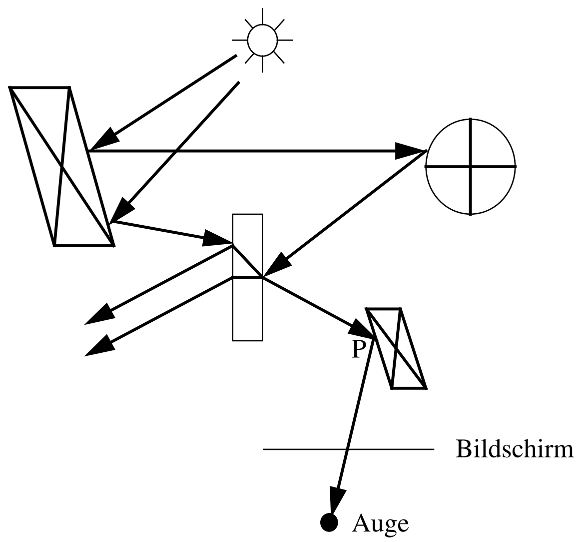

Abbildung 8: Prinzip der Strahlverfolgung

ten gute Ergebnisse in bezug auf den Speedup aufgrund des enormen Rechenzeitbedarfs des Ray-Tracing-Verfahrens.
Auch in *pray* wird deshalb die Bildraumaufteilung zeilenweise vorgenommen, d.h. jeder Netzwerk-Transputer berechnet eine Zeile des Bildes. Bei Beendigung der Rechnung wird die fertige Zeile an das GDS gesendet und zusätzlich eine Meldung an den Host-Transputer. Danach kümmert sich der Prozessor wieder um einen neuen Job, d.h. um eine neue Bildzeile.

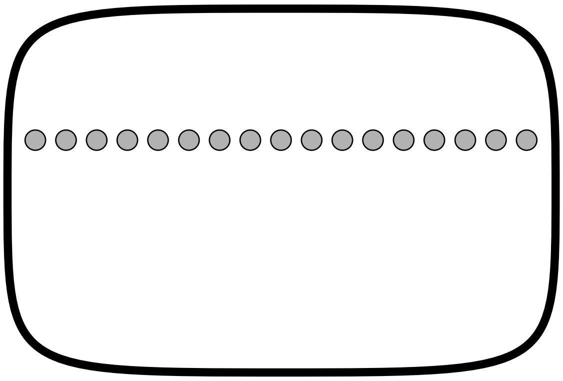

Abbildung 9: Zeilenaufteilung

Die Meldung an den Host-Transputer dient zur Buchhaltung der verschickten bzw. ausgeführten Jobs, so daß nach Beendigung aller Jobs die Terminierungsphase vom Host-Transputer eingeleitet werden kann. Der Host-Transputer ist verantwortlich für das Erzeugen der einzelnen Jobs, die dann ins Netzwerk gesendet werden und dort von den einzelnen Netzwerk-Transputern nach einem Verfahren untereinander aufgeteilt werden, das unabhängig von der Prozessortopologie ist, und im folgenden noch genauer beschrieben wird. Da Bilder, die mit dem Ray-Tracing-Verfahren erzeugt worden sind, ihre ganze Schönheit bzw. fotorealistische Qualität erst auf einem True-Color-Monitor entfalten, ist die Ausgabe von *pray* auch nur auf dem GDS vorgesehen.
Bei den ernormen Rechenzeiten der Ray-Tracing-Bilder ist es aber auch nötig, die fertigen Ergebnisse abspeichern zu können. Aus diesem Grunde kann *pray* nach Beendigung der Rechnung das Bild mit Hilfe von *grafik* abspeichern, um es z.B. unter dem X-Window-System zu verarbeiten oder um es zu einem späteren Zeitpunkt direkt wieder auf dem GDS anzeigen zu können. Alle Prozessoren im Netzwerk sind in *pray* die sogenannten Netzwerk-Transputer, die für die Berechnung des Bildes verantwortlich sind. Die gesamte Szeneninformation muß deshalb auf diesen Prozessoren vorhanden sein und wird deshalb sofort beim Aufruf des Prozesses als Parameter übergeben. Dies erzeugt natürlich einen gewissen Speicheroverhead, der aber aufgrund der besseren Auslastung der einzelnen Prozessoren gegenüber anderen Methoden in Kauf genommen werden kann.
Zusätzlich zum bisherigen Algorithmus ist in *pray* noch zu Beginn eine dynamische Spannbaumberechnung, mit Host-Transputer und GDS als Wurzeln, auf jedem Prozessor implementiert. Obwohl auch für *pray* im Verlauf der Entwicklung ein *deBruijn*-Netzwerk verwendet wurde, dessen kürzeste Wege einfach zu bestimmen sind (vgl. [Leig92]), hat diese Berechnung den Vorteil, daß sie netzwerkunabhängig ist. Auch hier fällt natürlich der zusätzliche Kommunikationsaufwand gegenüber der enormen Rechenzeit zur Berechnung der Bilder nicht belastend ins Gewicht. Die ansonsten eher einfache Parallelisierungsidee erfährt durch diese dynamischen Berechnungen der kürzesten Wege von jedem Prozessor im Netz zu den Wurzeln des Spannbaums eine Aufwertung, durch die eine universelle Einsetzbarkeit des Back-End *pray* erreicht wird.
Die naheliegendste Parallelisierung des Ray-Tracing-Verfahrens ist auch, von der Effizienz her betrachtet, die beste. Wie aus der Beschreibung schon zu vermuten ist, liegt, in der Phase der Berechnung, in der alle Prozessoren jeweils eine Zeile des Bildschirms in Bearbeitung haben, eine nahezu optimale Lastverteilung vor. Während der Endphase, in der es im schlimmsten Fall vorkommen kann, daß z.B. nur noch ein Prozessor eine ganze Zeile zu berechnen hat, während für alle anderen keine Jobs mehr vorhanden sind, kommt es natürlich zu Effizienzverlusten. Diese könnten eventuell durch eine feinere Aufteilung dieser Zeilen vermieden werden. Die mit *pray* gemachten Versuche und Zeitmessungen zeigen jedoch, daß der Aufwand für das Feststellen einer solchen Situation bzw. das Verteilen und Einsammeln der kleineren Zeilen-

stücke sich nicht notwendigerweise beschleunigend sondern sogar verlangsamend auswirkt. Da dies jedoch auch wieder stark szenenabhängig ist, kann im Mittel damit gerechnet werden, daß nur wenig Effizienzverlust auftritt, wenn dieses Phänomen unberücksichtigt bleibt. Die Startphase, in der ja auch nicht alle Prozessoren zur gleichen Zeit mit Arbeit versorgt sind, ist so kurz, daß sie ebenfalls nicht ins Gewicht fällt.

Allgemein wurden auch für dieses Verfahren stark unterschiedliche Bilder auf bis zu 64 Transputern berechnet. Es ergab sich eine mittlere Effizienz von etwa 90%, was auf eine gute Parallelisierung schließen läßt. Ein Teil des Effizienzverlusts ist bereits beschrieben worden. Der andere Teil ist die Folge des Kommunikationsoverheads, der wegen der dynamischen Berechnung der kurzen Wege entstanden ist, was aber durch die erhaltene Topologieunabhängigkeit bzw. universelle Einsetzbarkeit aufgewogen wird. Wird die Effizienz mit der von anderen parallelen Verfahren verglichen, so liegt sie im oberen Bereich.

Nichtsdestotrotz ist die Herstellung von Ray-Tracing-Bildern auch nach wie vor noch eine ziemlich aufwendige Angelegenheit. Die absolute Rechenzeit einzelner Bilder bewegt sich, auch trotz der Parallelisierung mit 64 Transputern, immer noch bei einer Auflösung von 800×600 Pixeln im Bereich von mehreren Minuten, so daß von interaktiver Anwendung des Ray Tracing im *parimod*-System keine Rede sein kann. Im Vergleich mit der Rechenzeit der sequentiellen Version kann aber die Wartezeit von mehreren Minuten schon als ziemlich kurz angesehen werden, mußte sonst zwischen Start und Ende der Berechnung auf Einprozessorsystemen meistens eine Nacht einkalkuliert werden. Eine besonders hohe Rechenzeit kann aber auch immer dann noch mit der parallelen Version erreicht werden, je größer der Anteil an spiegelnden und durchsichtigen Objekten in der Szene wird. Für große Bilder mit hoher Anzahl von Pixeln sollte das Ray-Tracing-Verfahren nach wie vor als high-quality Back-End angesehen werden, wobei zum Erstellen der Szene die Verfahren bzw. Parallelisierungen aus Abschnitt 3.1 verwendet werden sollten. Zum Abschluß kann das Bild dann mit dem Ray-Tracing-Verfahren dargestellt werden.

Da zusätzlich auch bei kleinen Bildgrößen (z.B. 256×256) Effekte, wie etwa Spiegelungen und Brechungen, gut beobachtet und schnell berechnet werden können, ist *pray* eine sinnvolle Ergänzung bzw. nicht mehr wegzudenkende Eigenschaft des *parimod*-Systems, die die Vorteile transputerbasierter Multiprozessorsysteme dokumentiert.

3.3 Das parallele Animationsprogramm *panim*

Mit dem parallelen Animationsprogramm *panim* (parallel animation tool) wird das *parimod*-System komplettiert (vgl. Abbildung 1). Es ist die konsequente Fortsetzung der parallelen Programme *pcg* und *pray*, in denen das Transputer-Netzwerk zur Parallelisierung der Berechnung einzelner Bilder benutzt wurde und ist ebenfalls in occam2 programmiert worden.

3.3.1 Topologie

Damit die Chancen einer Echtzeitanimation auf Transputern überhaupt realistisch eingeschätzt werden können, wurde zunächst eine genaue Analyse der zur Verfügung stehenden Hard- und Software (vgl. dazu [INM88a/b/c], [INM89], [PAR89] und [PAR90a/b])vorgenommen und mehrere mögliche Topologien umfangreich untersucht (vgl. [Zep93]).

Als Ergebnis läßt sich festhalten, daß die Überwindung des Flaschenhalses, der entsteht, wenn die fertig berechneten Bilder aus dem Netzwerk zum GDS übertragen werden sollen, nur durch eine ausgefeilte Programmierung erzielt werden kann, denn auf den Links kann eine Übertragungsgeschwindigkeit von maximal 20 MBits/Sek in beiden Richtungen erreicht werden, ohne die CPU nennenswert zu belasten. Diese Werte lassen hoffen, daß für Bilder in einer Auflösung von 256×256 Pixeln Bildfrequenzen erreichbar sind, die irgendwo zwischen den für die Echtzeitanimation gewünschten 24 Bildern und den mindestens benötigten 15 Bildern pro Sekunde liegen, denn ab einer solchen Frequenz nimmt das Auge die Bilder als Folge und nicht mehr als Einzelbilder war. Abbildung 10 zeigt die aus den Überlegungen resultierende Topologie, die im weiteren Verlauf als Zwölfer-Pipeline bezeichnet wird.

Um die hier angesprochene Transferrate aber auch wirklich zu erreichen, sollten die Links

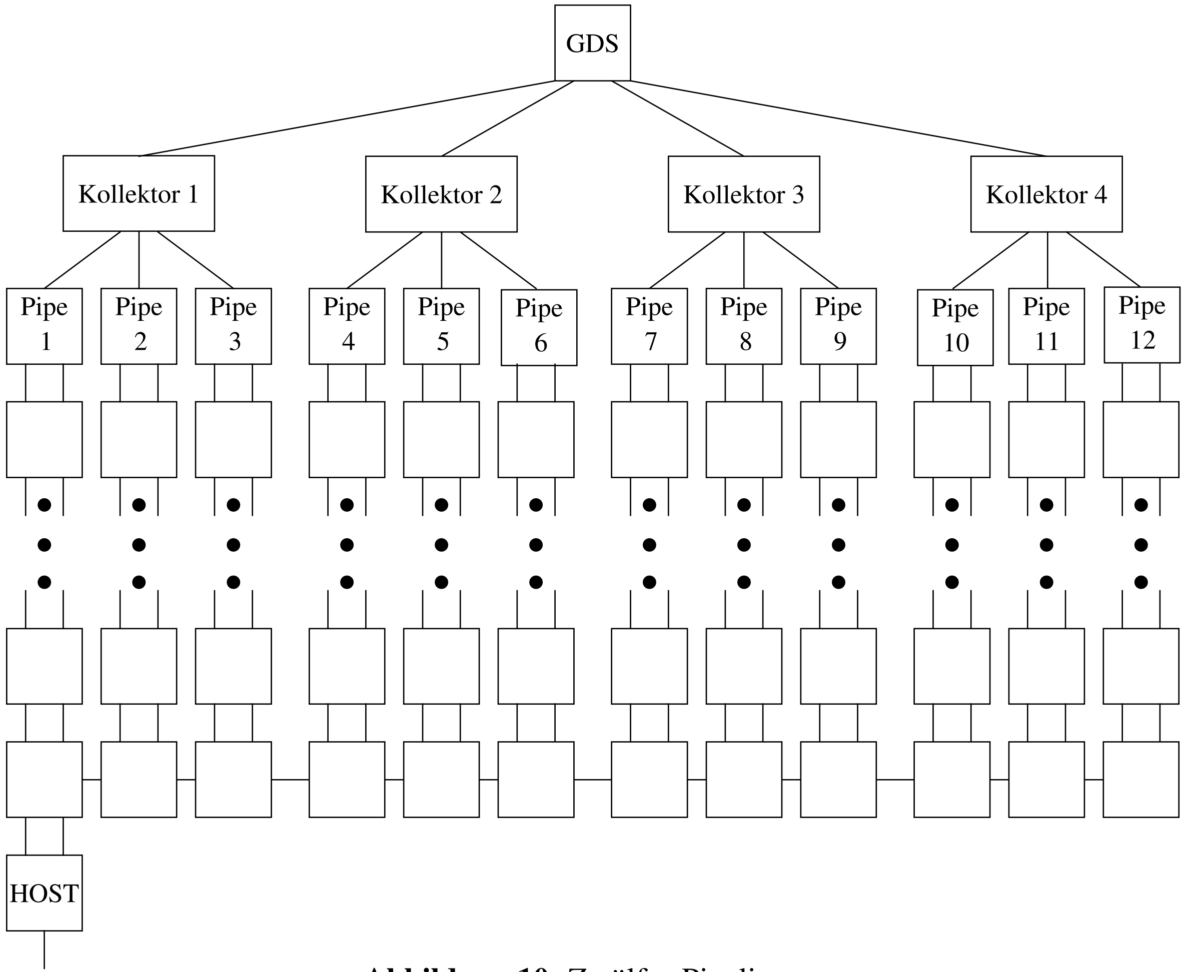

Abbildung 10: Zwölfer-Pipeline

viele, möglichst große Datenpakete übertragen, d.h. in Abbildung 10 müssen aus dem Netzwerk ständig Daten zum GDS-Transputer gesendet werden und dieser muß ständig über seine vier Links diese Daten empfangen.

Die Zahl der Pipelines, in der jeweils ein Bild der Animationssequenz berechnet wird, beträgt zwölf, und die Funktion der Prozessoren, die direkt mit dem GDS verbunden sind, ist die eines Zwischenpuffers für drei berechnete Bilder, die aus den einzelnen Pipelines aufgenommen werden können. Aus diesem Grunde werden diese Prozessoren als Kollektoren bezeichnet. Durch die Einführung dieser Kollektoren wird offensichtlich der Durchsatz an Bildern erhöht. Die Verbindung aller Pipelines mit dem Host-Transputer wird durch die freien Links der Prozessoren, die am Anfang jeder Pipeline plaziert sind, hergestellt. Auch diese Topologie ist skalierbar und nutzt alle Linkverbindungen aus.

Bei der Animation hat nun jede Pipeline nur noch jedes zwölfte Bild zu berechnen, wobei die drei Pipelines, die gemeinsam an einem der vier Kollektoren hängen, nicht jeweils aufeinanderfolgende Bilder berechnen, sondern um vier Bilder versetzte Berechnungen durchführen, wie es in Abbildung 11 schematisch dargestellt ist.

Durch diese geschickte Verteilung der Berechnungen der Einzelbilder auf die verschiedenen Pipelines kann das GDS weiterhin auf seinen Links der Reihe nach vier aufeinanderfolgende Bilder erhalten. Ebenso wird die Gefahr der ruckartigen Bildanzeige durch diese Einteilung weitestgehend vermieden, wenn die Berechnungszeiten für ein einzelnes Bild in jeder Pipeline klein gehalten werden können. Auf dem GDS entsteht durch viermaliges Abfragen der Links in der Reihenfolge von links nach rechts die Bildfolge i + 0 bis i + 11.

Der Berechnungsspielraum pro Bild, der in der Zwölfer-Pipeline für jedes Bild zur Verfügung steht, beträgt etwa eine halbe Sekunde, wenn 24 Bilder pro Sekunde erzeugt werden sollen. Doch die auftretenden Verluste durch die Kommunikation und die Bildanzeige verlangsamen

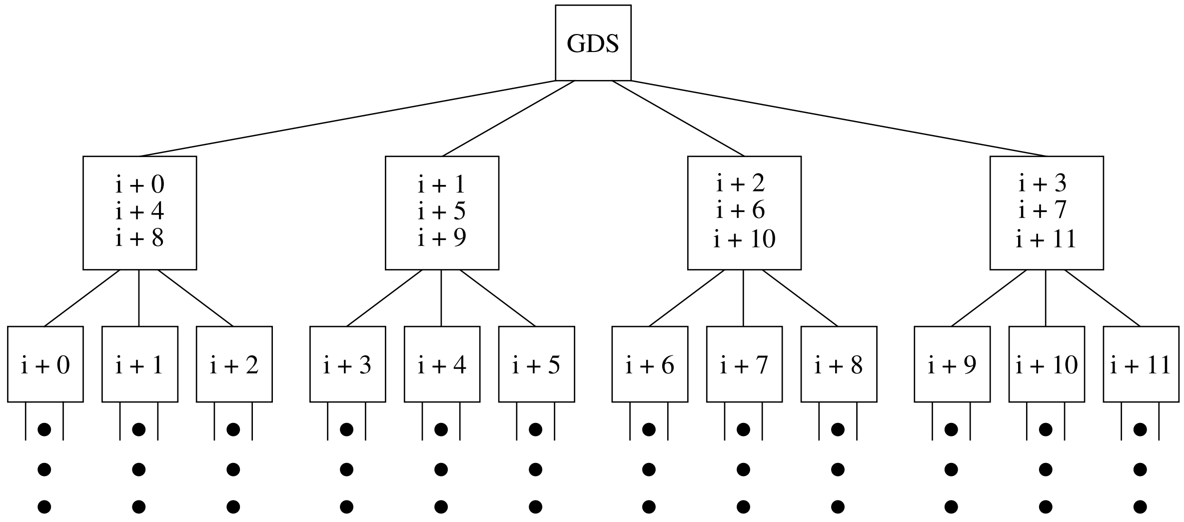

Abbildung 11: Einzelbildberechnung in Zwölfer-Pipeline

die Frequenz etwas, so daß mit dieser Topologie letztendlich 18 Bilder pro Sekunde in einer Auflösung von 256×256 Bildpunkten erreicht worden sind.
Durch die Skalierbarkeit der Zwölfer-Pipeline, die dadurch erreicht wird, daß aus der Mitte der zwölf Pipelines jeweils Prozessoren eingefügt oder weggenommen werden können, ist der Einsatz von *panim* auf unterschiedlich vielen Prozessoren möglich. Werden der GDS-Transputer und der Host-Transputer mitgezählt, so beginnt die kleinste Topologie bei 30 Prozessoren, da mindestens zwei Prozessoren in jeder Pipeline vorhanden sein müssen. Die volle Leistungsfähigkeit, auf dem für diese Arbeit zur Verfügung stehenden System, wurde jedoch erreicht, wenn in jeder Pipeline 5 Transputer zur Verfügung stehen und somit insgesamt 66 Transputer zur parallelen Berechnung eingesetzt werden.

3.3.2 Parallele Bildberechnung

Wie schon in den vorigen Abschnitten beschrieben, wird in jeder Pipeline genau ein Bild berechnet, wobei das Gouraud Shading und die z-Buffer-Technik zur Schattierung der Szenen verwendet werden. Die Beleuchtung der Szenen wird durch Progressive Refinement Radiosity (vgl. [GCT86] und [CCWG88]) iterativ auf dem Host-Transputer berechnet, und in Intervallen werden die Teilergebnisse auf die einzelnen Pipelines verteilt. Je länger sich also der Benutzer in der Szene aufhält, egal ob stehend oder fortbewegend, desto realistischer werden die Beleuchtungsverhältnisse dargestellt.
Die Aufteilung des z-Buffers geschieht in jeder Pipeline in Form einzelner horizontaler Streifen, wobei zu Beginn jeder Prozessor in der Pipeline die gleiche Anzahl von Zeilen erhält. Da aber diese statische Aufteilung zu Beginn und natürlich auch während der Animation zu stark unterschiedlich ausgelasteten Prozessoren führen kann und in der Regel auch führt, wird ein dynamischer Lastenausgleich zwischen den Prozessoren der einzelnen Pipelines während der Berechnung der Bilder durchgeführt. Dazu werden die Zeiten, die für die Berechnung der jeweiligen z-Buffer für das letzte Bild auf jedem Transputer der Pipeline benötigt wurden, genau wie die fertigen Bilder selbst, vom Anfang der Pipeline zum Ende gesendet. Dies geschieht mit Hilfe eines Prozesses und über die zweite Linkverbindung der Prozessoren in den Pipelines. Der Prozessor am Ende einer Pipeline berechnet nun den Mittelwert aus den Zeiten zur Berechnung der einzelnen z-Buffer und führt danach, falls es Prozessoren gibt, die von diesem Wert abweichen, eine neue Aufteilung der Grenzen bzw. der Zeilen durch. Danach ist die Last der einzelnen Prozessoren wieder gleichverteilt.
Zur Beschleunigung der Zeiten, die zur Berechnung eines Bildes benötigt werden, sind einige Verfahren zur Vorberechnung nicht sichtbarer Flächen und das Halbbild-Verfahren implementiert und eingesetzt worden. Beim Halbbild-Verfahren werden von Bild zu Bild nur die ungera-

den bzw. die geraden Zeilen berechnet. Durch die Frequenz von 18 Bildern bei der Anzeige wird dann jeweils ein Bild aus zwei aufeinanderfolgenden Halbbildern zusammengesetzt.
Durch den Einsatz dieser Verfahren zeigte sich jeweils eine deutliche Verringerung des Berechnungsaufwands, so daß ihr Einsatz berechtigt ist. Folglich wurde es möglich, Wohnraumszenen, die aus bis zu 1000 Flächen (Patches) bestehen können, in der erforderlichen Zeit zu berechnen, bei einer Bildfrequenz von 18 Bildern pro Sekunde. Abbildung 12 zeigt die Einsicht in einen Wohnraum, der mit *panim* durchschritten werden kann.

3.3.3 Animationsarten

Insgesamt bietet *panim* dem Benutzer drei verschiedene Arten der Animation. Die erste Art, die mit *panim* durchgeführt werden kann, ist die Animation der Bewegung einer Kamera durch eine Szene. Dazu liest *panim* eine von *ism* erstellte Szenenbeschreibungsdatei ein und verteilt die Informationen, die in dieser Datei enthalten sind, auf die einzelnen Netzwerk-Transputer. Danach wird in jeder Pipeline die Berechnung der Einzelbilder durchgeführt, und die Ergebnisse werden mit Hilfe der Kollektoren zum GDS gesendet und dort angezeigt. In allen Pipelines wird zu Beginn das gleiche Bild erzeugt und auch vom GDS jeweils angezeigt, da kein Befehl zur Bewegung vorliegt. Die Befehle zur Bewegung der Kamera werden on line von der Tastatur des Host-Rechners eingegeben, wobei der gegenwärtige Standort und die Blickrichtung immer durch den Normal Reference Point bzw. die Richtung von diesem Punkt zum View Reference Point definiert sind.
Eine von der Implementation im wesentlichen identische Variante der Animation bietet die Flugsimulation. Die Verhaltensweise der Flugsimulation ist eine permanente Vorwärtsbewegung in Blickrichtung, die der Benutzer nun durch Steuerung per Tastendruck in die gewünschten Bahnen lenken kann.
Das Bewegen von einzelnen Objekten durch eine Szene stellt die dritte Art der Animation von *panim* dar. Damit ist gemeint, daß z.B. von einem festen Kamerastandpunkt aus die Bewegung eines bisher feststehenden Objekts in der Szene animiert wird bzw. gesteuert werden kann.
Da diese Kombination der verschiedenen Arten der Animation sehr reizvoll ist und eine natürliche Fortsetzung von Kamerafahrt und Flugsimulation darstellt, ist sie ebenfalls als letzte Funktion in Ansätzen in *panim* implementiert worden und auf ihre Durchführbarkeit getestet worden. Probleme ergeben sich allerdings bei der Beleuchtung der Szene, da sich jetzt die Szenengeometrie während der Animation verändert.

3.3.4 Ergebnisse

Wie schon bei den bisher implementierten parallelen Programmen *pcg* und *pray* sind die Ergebnisse, die bei der Echtzeitanimation auf Transputern mit *panim* erzielt werden können, stark abhängig von der Komplexität der Szene. War diese Eigenschaft bei den Programmen *pcg* und *pray* in Form von stark unterschiedlichen Laufzeiten zu beobachten, wobei natürlich komplexere Szenen erheblich längere Rechenzeiten in Anspruch nahmen als einfache Szenen, so steht und fällt aber der Anspruch der Echtzeitanimation bei *panim* insbesondere mit der Komplexität der Szene. Nur wenn in jeder Pipeline des Transputer-Netzwerks aus Abbildung 10 mindestens 1.5 Bilder pro Sekunde erzeugt werden können, so sind die dadurch erzielten 18 Bilder pro Sekunde auch darstellbar.
Trotz dieser, für universelle Prozessoren, erheblichen Einschränkungen an die Rechenzeit, können aber mit *panim* sowohl Kameraanimation als auch Flugsimulation in Innenräumen durchgeführt werden, die eine Komplexität aufweisen, wie die Beispiele in den Abbildungen 12 und 13 zeigen. Dies wird zum größten Teil durch das angewandte Halbbild-Verfahren und die, speziell für diese Topologie entworfene, dynamische Lastverteilung erreicht. Hält sich also der Benutzer an Szenen, in denen etwa 30 Objekte plaziert sind, die für das Progressive Refinement Radiosity in bis zu 1000 kleine Flächen (Patches) unterteilt werden, so kann mit jeweils fünf Prozessoren pro Pipeline diese Bildfrequenz erreicht werden.
Alles in allem bietet *panim* jedoch eine qualitativ sehr gute Animation, die bewußt für dieses

Transputer-System implementiert wurde, um die sich bietenden Möglichkeiten konsequent auszunutzen und dabei aufzuzeigen, was damit machbar ist. Daß auch schon für ein solch kleines System eine gute Animation berechnet werden kann, bewahrt davor immer auf noch mögliche Verbesserungen oder Erweiterungen verweisen zu müssen.
Somit bietet *panim* dem Benutzer, im Rahmen des *parimod*-Systems, ein umfangreiches Werkzeug zur Herstellung und Nutzung von Echtzeitanimation auf universellen Multiprozessor-Systemen.

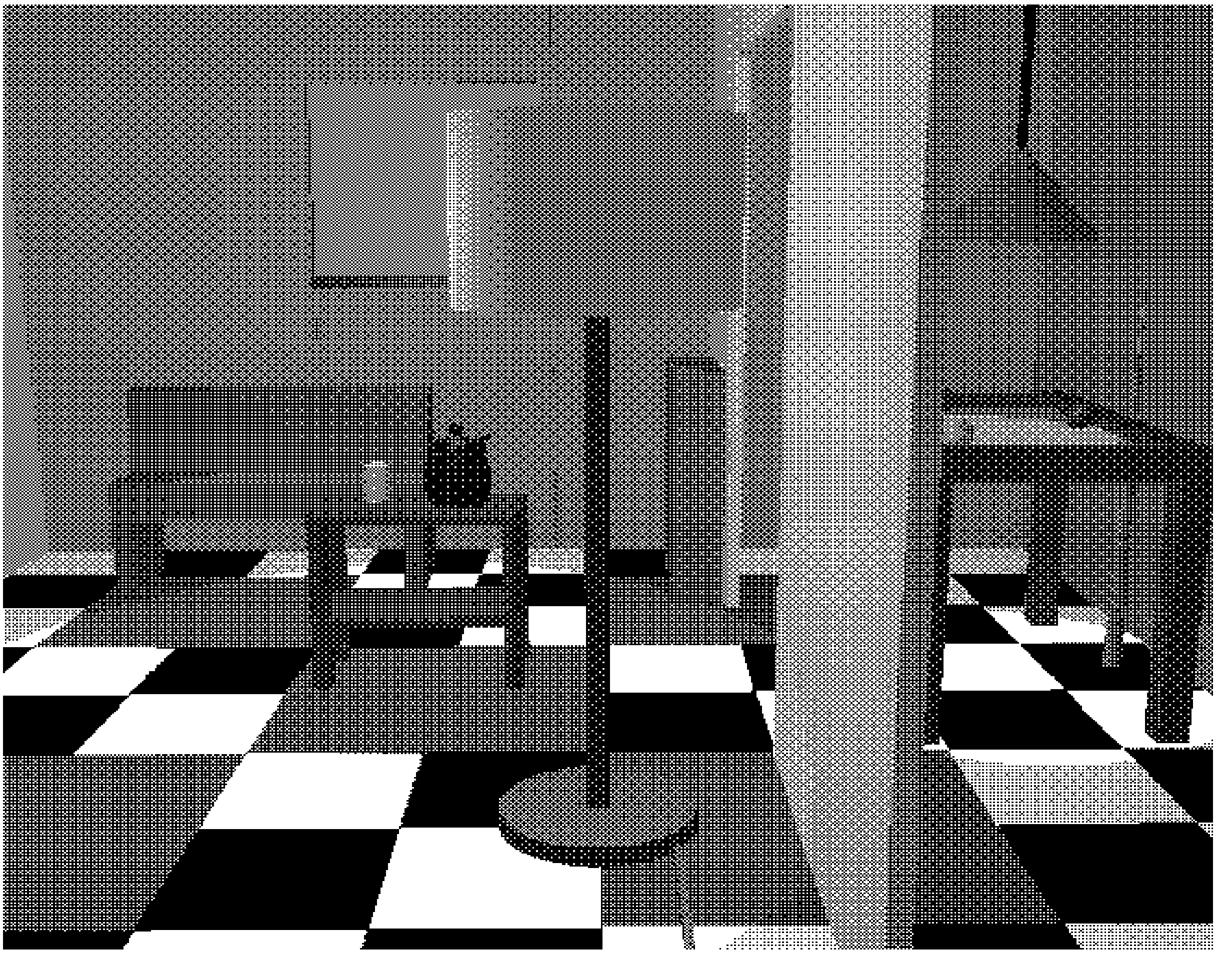

Abbildung 12: *parimod*-Beispielszene in Graustufen

4. Zusammenfassung und Ausblick

Die Berechnung von fotorealistischen Darstellungen dreidimensionaler Szenen erfordert einen extrem hohen Rechenzeitbedarf und wird dadurch zu einem natürlichen Kandidaten für die Parallelverarbeitung. Obwohl heutzutage immer mehr Systeme, in denen teure Spezialhardware zum Einsatz kommt, speziell zur Lösung dieser Anforderungen gebaut werden, eignen sich aber auch universelle Multiprozessor-Systeme ohne spezielle Graphikeigenschaften aufgrund ihrer Flexibilität und ihrer enormen Rechenleistung für solche Anwendungen.
Das in diesem Artikel vorgestellte *parimod*-System ist der Beweis für diese Behauptung. Die in den vorherigen Abschnitten vorgestellten Ergebnisse zeigen, daß dieser Ansatz einer Verbindung von Parallelverarbeitung und Computergraphik als Beispiel einer Anwendung durchaus zu zufriedenstellenden Resultaten führt. Die Behinderung durch fehlende Spezialhardware kann in weiten Teilen durch die Leistungsfähigkeit des Zusammenschlusses beliebig vieler Prozessoren ausgeglichen werden. Durch den Entwurf effizienter paralleler Methoden, die dann auf diesen Systemen eingesetzt werden, kann ein solcher Nachteil weitestgehend vermindert werden. Im Fall des Ray Tracing können sogar auch erhebliche Verbesserungen erreicht werden. Auch die Ergebnisse aus dem Bereich der Computergraphik, insbesondere der Echt-

zeitanimation, sind gut und ein weiterer Beweis für die große Flexibilität universeller Multiprozessorsysteme.

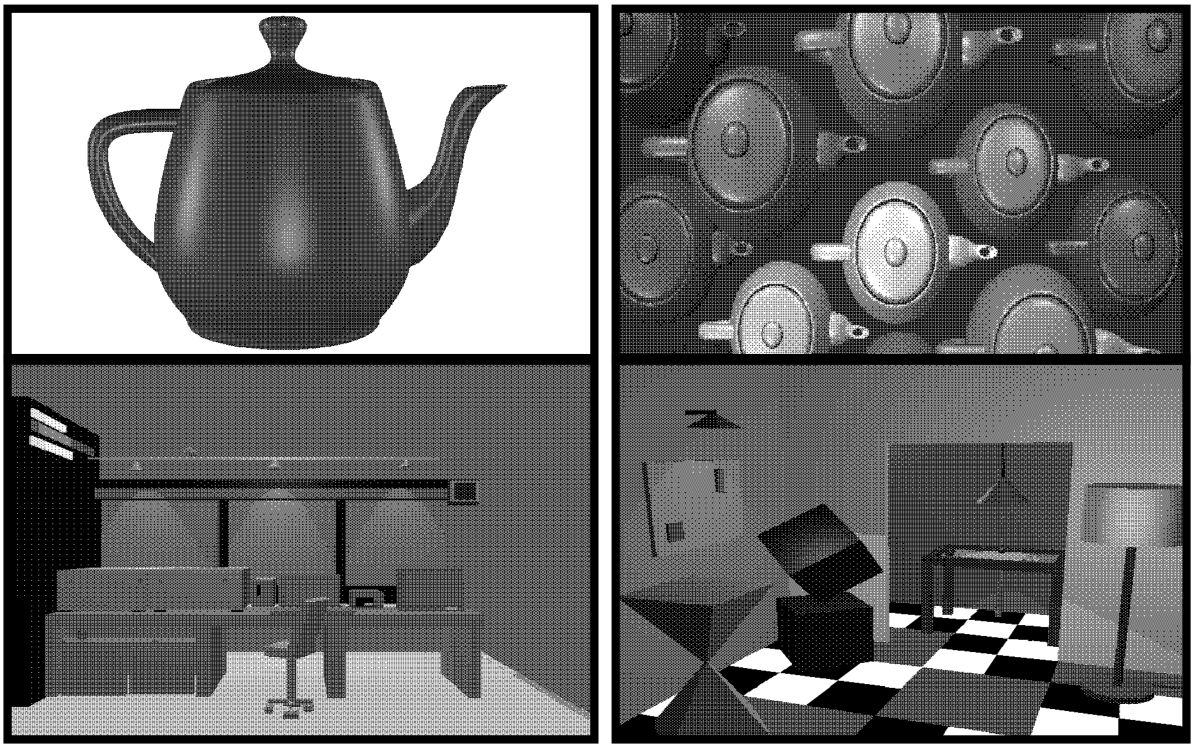

Abbildung 13: *parimod*-Beispielszenen in Graustufen

Das *parimod*-System als Ganzes gesehen liefert somit ein flexibles Beispiel für den Einsatz von transputerbasierten Multiprozessor-Systemen. Insbesondere in der Architektur oder im Bereich der Animation und Visualisierung, z.B. von Molekülen in der Biologie oder Chemie, können mögliche Einsatzgebiete dieses Systems liegen.
So rasant wie die Weiterentwicklung der Parallelverarbeitung und der Computergraphik sich zur Zeit vollzieht, so unvollständig muß das hier vorgestellte System bzw. die durchgeführte Forschung zu diesen beiden Themen bleiben. Die Weiterentwicklung von Hardware ermöglicht es mittlerweile schon mit acht oder mehr Links, Verbindungen vom GDS ins Transputer-Netzwerk zu schalten. Verbindungen vom Host-Transputer zum Host-Rechner sind schneller geworden und haben größere Kapazitäten, so daß auch die Anzeige der Bilder auf dem Host-Rechner komfortabler und schneller bewältigt werden kann. Da aber die erzielten Ergebnisse bewußt auf dem hier zur Verfügung stehenden System gelassen und auch erreicht worden sind, können diese Weiterentwicklungen für die bisher erzielten guten Resultate nur im positiven Sinne zur Verbesserungen führen.
Aus dem weiten Feld der Computergraphik ist zwar schon in dieser Arbeit ein weites Spektrum integriert worden, trotzdem sind aber Themenbereiche, wie etwa das Antialiasing oder das Texture Mapping, noch nicht behandelt worden und müssen somit in den Bereich der weiteren Forschung aufgenommen werden.
Auch der Bereich der interaktiven Szenenmodellierung ist natürlich noch nicht vollständig in diesem System erfaßt worden. Eine weitere sinnvolle Ergänzung, die die interaktive Szenenmodellierung noch erweitern und verbessern würde, ist die Zusammenfassung bzw. Gruppierung von einzelnen Basisobjekten zu einem neuen Objekt, so daß z.B. einzelne Operationen jeweils auf diesem neuen Objekt nur einmal anzuwenden sind.
Allgemeines Fazit ist jedoch, daß die hier vorgestellte parallele Computergraphik und Animation auf Transputern, zusammengefaßt im *parimod*-System, auf der zur Verfügung stehenden Hardware die zu Beginn dieses Artikels gestellten Anforderungen und Ziele zufriedenstellend erfüllt.

Literatur:

[BaRu84] M. BANAHAN, A. RUTTER, *UNIX lernen, verstehen, anwenden,* Carl Hanser Verlag, 1984

[BoGi82] J.W. BOYSE, J.E. GILCHRIST, *GMSolid: Interactive Modeling for Design and Analysis of Solids*, IEEE, Computer Graphics & Applications 2(2), pp. 27- 40, 1982

[CCWG88] M.F. COHEN, S.E. CHEN, J.R. WALLACE, D.P. GREENBERG, A Progressive Refinement Approach to Fast Radiosity Image Generation, Computer Graphics, Vol. 22, No. 4, Aug. 1988, pp. 75 - 84

[Cro87] F. CROW, *The Origins of the Teapot*, IEEE Computer Graphics & Applications, Vol. 7, No. 1 (Januar 1987), pp. 8 - 19

[FDFH90] J.D FOLEY, A. VAN DAM, S.K. FEINER, J.F. HUGHES*, Computer Graphics: Principles and Practice; Second Edition*, Addison-Wesley, 1990

[GCT86] D.P. GREENBERG, M.F. COHEN, K.E. TORRANCE, Radiosity: A Method for Computing Global Illumination, The Visual Computer, Vol. 2, No. 5, Sept. 1986, pp. 291-297

[Gla89] A.S. GLASSNER, *An Introduction to Ray Tracing*, Academic Press, 1989

[Gre91] A.S. GREEN, *Parallel Processing for Computer Graphics*, The MIT Press, Cambridge MA, 1991

[HHHW91] T.L.J. HOWARD, W.T. HEWITT, R.J. HUBBOLD, K.M. WYRWAS, *A Practical Introduction to PHIGS and PHIGS PLUS*, Addison-Wesley, 1991

[INM88a] INMOS LIMITED, *Occam2 Reference Manual*, Prentice Hall, 1988

[INM88b] INMOS LIMITED, *Transputer Development System*, Prentice Hall, 1988

[INM88c] INMOS LIMITED, *Transputer Reference Manual*, Prentice Hall, 1988

[INM89] INMOS LIMITED, *Transputer Technical Notes*, Prentice Hall, 1989

[JoGo88] G. JONES, M. GOLDSMITH, *Programming in Occam2,* Prentice Hall, 1988

[KeRi83] B.W. KERNIGHAN, D.M. RITCHIE, *Programmieren in C*, Carl Hanser Verlag, 1983

[Leig92] F.TH. LEIGHTON, *Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*, Morgan Kaufmann Publishers, 1992

[Mänt88] M. MÄNTYLÄ, *An Introduction to Solid Modeling*, Computer Science Press, 1988

[Mor85] M.E. MORTENSEN, *Geometric Modeling*, John Wiley & Sons, 1985

[ORei90a] T. O'REILLY, *X Window System, Volume 1: Xlib Programming Manual; Second Edition*, O'Reilly & Associates, 1990

[ORei90b] T. O'REILLY, *X Window System, Volume 2: Xlib Reference Manual; Second Edition*, O'Reilly & Associates, 1990

[PAR89] PARSYTEC GmbH, *Multi Tool 5.0 Technical Documentation - Transputer Programming Environment*, 1989

[PAR90a] PARSYTEC GmbH, *MultiCluster-2 Technical Documentation - Installation, Expansion and Maintenance Manual*, 1990

[PAR90b] PARSYTEC GmbH, *GDS-2 Graphic Display Subsystem*, 1990

[WaWa92] A. WATT, M. WATT, *Advanced Animation and Rendering Techniques*, Addison-Wesley, 1992

[Zep93] K. ZEPPENFELD, *Parallele Computergraphik und Animation mit Transputern*, DeutscherUniversitätsVerlag, 1993